IN1000 Innleveringsoppgave 6

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) 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: Motorsykkel

Filnavn: *motorsykkel.py* og *test_motorsykkel.py*

Du skal skrive en klasse, *Motorsykkel,* som skal modellere kjøring av motorsykler. 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) og de instansvariablene klassen trenger. Instansvariabelen for kilometerstand skal alltid ha startverdi 0. De to andre instansvariablene skal initialiseres med verdier fra init-metodens parametere.

2. Skriv en metode, *kjor(self, km),* som øker kilometerstanden med det gitte antall kilometer (*km*).

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

4. Du skal nå begynne på et testprogram for klassen *Motorsykkel*. Opprett en ny fil, *test_motorsykkel.py*. Øverst i filen skal du importere klassen *Motorsykkel*. Deretter skal du skrive en foreløpig tom prosedyre, *hovedprogram()*, og kalle på denne. Kjør testprogrammet og rett opp eventuelle feil.

5. Inni *hovedprogram()* skal du nå opprette et objekt av klassen *Motorsykkel*, med et merke og et registreringsnummer som du velger selv. Opprett så to objekter til på samme måte. Kjør testprogrammet og rett opp eventuelle feil.

6. Utvid klassen *Motorsykkel* med en metode, *skriv_ut(self),* som skriver ut merke, registreringsnummer og kilometerstand. Utvid deretter prosedyren *hovedprogram* i testprogrammet, med et kall på metoden *skriv_ut* på hvert av objektene du har laget. Kjør testprogrammet og rett opp eventuelle feil.

7. I testprogrammet skal du nå øke kilometerstanden på den motorsykkelen du laget sist med 10 km, og deretter skrive ut resultatet av et kall på *hent_kilometerstand* i terminalen. Kjør testprogrammet og rett opp eventuelle feil. Kontroller spesielt at kilometerstanden ble oppdatert riktig.

8. Frivillig/bonus: Legg til *assert*-setninger i testprogrammet ditt (i prosedyren *hovedprogram*) etter hvert metodekall, slik at du vil få en feilmelding (*AssertionError*) dersom resultatet av metodekallet ikke er som forventet. Metoder som returnerer en verdi kan kalles direkte i *assert*-setningen - for eksempel slik:
<rett etter at et Motorsykkel-objekt er opprettet for variabelen mc>
assert mc.hent_kilometerstand() == 0

Syntes du denne oppgaven var vanskelig? Se Trix-oppgave 7.01.
Syntes du denne oppgaven var enkel? Se Trix-oppgave 7.03.

Oppgave 2: Teorioppgave

Filnavn: *teori.txt*

Gi korte svar på spørsmålene under (hentet fra ukens forelesning):

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 (ofte kalt bare metode)? Hvordan skiller metoder seg fra prosedyrene/funksjonene vi har jobbet med hittil?

Oppgave 3: Hund

Filnavn: *hund.py* og *test_hund.py*

1. Skriv en klasse, *Hund*. Konstruktøren skal ha parametere som angir initialverdier (startverdier) til instansvariablene *alder* og *vekt*. *Hund* skal i tillegg ha en instansvariabel *metthet*, som alltid initialiseres med verdien 10.

2. Lag et testprogram i filen *test_hund.py,* slik at du kan teste klassen din etter hvert som du utvider den. Bruk designet fra oppgave 1 (legg alle testene i en prosedyre *hovedprogram*, som du må huske å kalle på). Foreløpig skal du kun opprette et objekt i testprogrammet, og deretter kjøre det.

   Endelig testprogram, slik det skal leveres, er beskrevet i punkt 6. Legg inn kall på hver metode du skriver og kjør testprogrammet underveis. Slik finner du feilene med

en gang de oppstår. Om du ønsker kan du erstatte print-setningene i testprogrammet med assert-setninger.

3. Skriv metodene *hent_alder*() og *hent_vekt*() som henter ut (returnerer) henholdsvis *alder* og *vekt*.

4. Skriv en metode, *spring,* uten parametere (kun self). Metoden skal redusere *metthet* med 1. Dersom *metthet* blir mindre enn 5, skal metoden redusere *vekt* med 1.

5. Skriv en metode, *spis*. Metoden skal ha en parameter (et heltall) som angir mengde og som skal legges til *metthet*. Dersom *metthet* blir større enn 7, skal metoden også øke *vekt* med 1.

6. *hovedprogram*-prosedyren i *test_hund.py* skal opprette et hundeobjekt, kalle på metodene *spring* og *spis* minst 2 ganger hver samt skrive ut hundens *vekt* etter hvert metodekall.

Syntes du denne oppgaven var vanskelig? Se Trix-oppgave 7.02, 7.05.
Syntes du denne oppgaven var enkel? Se Trix-oppgave 7.06.

Oppgave 4: Dato

Filnavn: *dato.py* og *test_dato.py*

1. Skriv en klasse, *Dato,* som representerer en dato. Konstruktøren til klassen skal ha parameterene *ny_dag*, *ny_maaned* og *nytt_aar*, og bruke disse til å initialisere instansvariabler i det nye objektet (dag, maaned og aar). Disse er heltall.

2. Begynn å skrive et testprogram i filen *test_dato.py,* på samme måte som i de tidligere oppgavene. Foreløpig trenger du bare å opprette et *Dato*-objekt for en dato der dag er 15. For hver nye metode du skriver videre, bør du legge inn kall på metoden i testprogrammet. For hvert kall, se om du får forventet resultat, for eksempel ved hjelp av kode i testprogrammet slik det skal leveres (beskrevet nederst i oppgaven). Spesielt for de frivillige metodene kan det være lurt å teste med flere forskjellige verdier for instansvariablene.

3. Utvid klassen med metoder som utfører operasjonene beskrevet under på en dato. Velg parametere til hver metode og om metoden skal returnere en verdi etter hva du synes er mest hensiktsmessig dersom det ikke er oppgitt. Begrunn valgene dine hvis du er usikker på hva oppgaven ber om.
    a. En metode *hent_aar*
    b. En metode som lager og returnerer en tekststreng som representerer datoen. Strengen skal være på en form som er grei å lese for mennesker, for eksempel “4.10.24”
    c. En metode (med en heltall-parameter *dag_nr)* som sjekker om dagen i datoen er *dag_nr* og returnerer True eller False
    d. Frivillig: En metode *ny_dag*,som endrer datoen i objektet til neste dag (kommenter eventuelle spesialtilfeller/ulike datoer metoden din ikke virker for)

e. Frivillig: En metode, *kommer_foer*, som tar inn *annen_dato* (en streng på formen “4.10.24”) og sjekker om denne datoen er før dato-objektet metoden kalles på. Metoden skal returnere True (hvis datoen i parameteren er tidligere enn datoen i objektet) eller False (hvis datoen i parameteren er den samme eller senere enn datoen i objektet).

Testprogrammet ditt bør minimum inneholde det som står i punktene nedenfor. Bruk metodene i Dato-objektet du opprettet (gjør egne valg av verdier og annet ved behov).

- Skrive ut datoens år på terminalen
- Lagre datoen i en variabel
- Skrive ut “Loenningsdag!” på terminalen hvis dag er 15, “Ny måned, nye muligheter” om dag er 1.
- Skrive ut datoen på terminalen på en lesbar form
- (Hvis du har gjort 2d: Endrer datoen til dagen etter og skriver den ut på nytt)
- (Hvis du har gjort 2e: Les inn en ny dato fra bruker og skriv ut på terminalen om den nye datoen kommer før den datoen objektet ditt representerer)

Syntes du denne oppgaven var vanskelig? Se Trix-oppgave 7.02, 7.05.
Syntes du denne oppgaven var enkel? Se Trix-oppgave 7.04, 7.09.

Oppgave 5: Viktige begreper i objektorientert programmering

Filnavn: *begreper.txt*

Se illustrasjonseksempelet nedenfor om du er i tvil om hvordan oppgaven skal besvares.

For hvert av begrepene i listen under, oppgi et linjenummer i programmet der du finner et eksempel på begrepet. Hvert begrep er gjennomgått i uke 7. Hvis eksempelet går over flere linjer, oppgir du start- og sluttlinje.

A. Klassedefinisjon
B. Opprettelse av nytt objekt
C. Definisjon av instansvariabel
D. Definisjon av instansmetode
E. Definisjon av konstruktør
F. Kall på instansmetode
G. Argumenter

```python
class Menneske:
    def __init__(self, navn, hoeyde, vekt):
        self._navn = navn
        self._hoeyde = hoeyde
        self._vekt = vekt

    def spiser(self, mat, antKalorier):
        print(self._navn + " spiser " + mat + ".")
        self._vekt += antKalorier / 8000  #ca. antall kalorier i en kg kroppsfett

    def sover(self, timer):
        print(self._navn + " sover " + str(timer) + " timer.")
        self._vekt -= (timer * 75) / 8000 #ca 75 kalorier forbrent per timer

    def veie(self):
        print(self._navn + " veier " + str(self._vekt) + " kg.")

print()
per = Menneske("Per",175,80)
paal = Menneske("Paal",180,90)
per.veie()
paal.veie()

print()
per.spiser("ribbe",1000)
paal.spiser("surkaal",300)
per.veie()
paal.veie()

print()
per.sover(8)
paal.sover(7)
per.veie()
paal.veie()

```

Illustrasjonseksempel

Oppgave:

For hvert av begrepene i listen under, oppgi et linjenummer i programmet der du finner et eksempel på begrepet. Hvis eksempelet går over flere linjer, oppgir du start- og sluttlinje.

A. variabeldeklarasjon
B. utskriftsetning
C. innlesing fra terminal
D. beslutning med flere alternativer

```
1 tall = int(input("Oppgi et tall: "))
2 if tall < 10 :
3     print ("Tallet er mindre enn 10")
4 else :
5     print ("Tallet er 10 eller høyere")
6
7
```

Mulige riktige svar på spørsmålene:

A. linje 1
B. linje 3
C. linje 1
D. linje 2-5

Oppgave 6: 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 og oppretter og initialiserer instansvariabler med disse. I tillegg skal konstruktøren opprette en instansvariabel, *hobbyer,* som en tom liste . Skriv en metode, *legg_til_hobby,* som tar i mot en tekststreng og legger den til i *hobbyer*-listen. Skriv også en metode, *skriv_hobbyer*. Denne metoden skal skrive ut alle hobbyene etter hverandre på en linje. Gi deretter Person-klassen en metode, *skriv_ut*, som i tillegg til å skrive ut navn og alder kaller på metoden *skriv_hobbyer*.

   Lag deretter et testprogram for klassen Person, der du lar brukeren skrive inn navn og alder, og oppretter 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 informasjon om brukeren skrives ut.

2. Løs oppgaven! Du skal levere **både oppgaveteksten (enten forslaget gitt i blått ovenfor eller en du lager selv) og besvarelsen**. Skriv oppgaveteksten som kommentarer over løsningen din. Oppgaven skal være et eget arbeid, og skal skille seg fra tidligere egenoppgaver og andre obligatoriske oppgaver.

Krav til innlevering

- Kun *.py*-filene og eventuelle *.txt*-filer 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 gi utskrifter som er enkle å forstå for brukere.

Hvordan levere oppgaven

1. Kommenter på følgende spørsmål i kommentarfeltet i Devilry. Spørsmålene **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) oppgave 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?
        iv. Hva vil du ha tilbakemelding på?

2. Logg inn på Devilry.

3. Lever alle *.py*-filene, *.txt*-filene som svarer på teorioppgavene, og *.txt*-filene som trengs for å kjøre programmene, 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.