IN1000 Obligatorisk innlevering 6

Frist for innlevering: 6. oktober kl 23.59 **Sist endret** 01.10.20

Introduksjon

Innleveringen består av 6 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: Motorsykkel

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

Du skal skrive en klasse *Motorsykkel* som skal modellere kjøringen 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) som initierer de instansvariablene klassen trenger.

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

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()*.

6. Inne i *hovedprogram* skal du opprette et objekt av klassen *Motorsykkel* med et merke, et registreringsnummer og en kilometerstand. Opprett så to objekter til på

samme måte. 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 7.01
Synes 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, og hvordan skiller dette seg fra prosedyrene/funksjonene vi har møtt hittil?

Oppgave 3: Hund

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

1. Skriv en klasse *Hund* med en konstruktør som tar imot verdier, 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. Sjekk 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 test-programmet ditt.

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

Oppgave 4: Dato

Filnavn: *dato.py* og *testDato.py*

1. Skriv en klasse Dato med som representerer en dato. Konstruktøren til klassen skal ha parametere nyDag, nyMaaned og nyttAar og bruke disse til å initialisere tilsvarende instansvariabler i det nye objektet (dag, maaned og aar) representert som heltall (integers).
2. Utvid klassen med metoder som utføre følgende operasjoner på en dato. Velg parametere til hver metode og om den skal returnere en verdi etter hva du synes er mest hensiktsmessig. Begrunn valgene dine om de ikke er opplagte.
    a. En metode som leser av året for datoen
    b. En metode som lager en streng (brukervennlig) av datoen
    c. En metode som sjekker om datoen er en bestemt dag i måneden
    d. Frivillig: En metode som leser inn en ny dato og sjekker om den er før eller etter
    e. Frivillig: En metode som endrer datoen til neste dag (velg og kommenter hvor mange spesialtilfeller/ ulike datoer metoden din virker for)
3. Skriv et testprogram som gjør følgende:
    a. Oppretter et objekt for en dato der dag er 15
    b. Skriver ut datoens år på terminalen
    c. Skriver ut “Loenningsdag!” på terminalen om dag er 15, “Ny maaned, nye muligheter om dag er 1.
    d. Lagrer datoen som en streng (på en form som er grei å lese for mennesker)
    e. Skriver ut datoen på terminalen på en lesbar form
    f. (Hvis du har gjort 2e: Endrer datoen til dagen etter og skriver den ut på nytt)
    g. (Hvis du har gjort 2d: Les inn en ny dato fra bruker og skriv ut på terminalen om den nye datoen kommer før, etter eller er lik den datoen objektet ditt representerer)

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

Oppgave 5: Objektorientert programmering: Viktige begreper

Filnavn: *begreper.txt*

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

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

```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()

```

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

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.

```
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
```

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

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 med 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).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.

- Programmet skal inneholde gode 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 lever/add delivery 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.