Informasjon

UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Skriftlig eksamen i IN1000
2025 HØST
Varighet: 5.12.25, 15:00 til 5.12.25, 19:00
Tillatte hjelpemidler: Alle trykte og skrevne, ingen digitale

Det er viktig at du leser hele forsiden før du begynner å besvare eksamensoppgavene.

Om du ønsker å zoome i oppgavesettet, hold ctrl nede og trykk + eller - på numerisk tastatur.

Faglærer vil besøke lokalet en gang mellom 16 og 18.

Eventuelle meldinger om feil eller presisering av oppgaver vil bli varslet med bjelle-ikonet øverst til høyre.

Dersom du mener en programmeringsoppgave er uklar, kan du gjøre egne forutsetninger og beskrive disse.

Du kan veksle mellom ulike språk (bokmål, nynorsk eller engelsk) øverst på siden.

Oppgavene gir til sammen 100 poeng. Poengfordelingen sier noe om vektingen av deloppgaver i sensuren.

1 Oppgave 1a (1p)

Hva skrives ut når følgende kode kjøres?

```
x = "3"
y = 4
x = int(x) + y
print(x)
```

Maks poeng: 1

2 Oppgave 1b (1p)

Hva skrives ut når følgende kode kjøres?

```
n = 10
if n > 5:
    n = n - 2
elif n > 7:
    n = n + 3
print(n)
```

Maks poeng: 1

3 Oppgave 1c (1p)

Hva skrives ut når følgende kode kjøres?

```
a = 5
if a > 3:
    a = a - 1
    if a == 4:
        a = a * 2
    else:
        a = 5
print(a)
```

Maks poeng: 1

4 Oppgave 1d (2p)

Hva skrives ut når følgende kode kjøres?

```
liste = [1, 2, 3, 4]
total = 0
i = 0
aktiv = True
while i < len(liste) and aktiv:
    total += liste[i]
    if total > 4:
        aktiv = False
    i += 1
print(total)
```

Maks poeng: 2

5 Oppgave 1e (1p)

Hva skrives ut når følgende kode kjøres?

```
tall = [5, 12, 20]
funnet = False
dusin = 12

for t in tall:
    if t == dusin:
        funnet = True
    else:
        funnet = False

if funnet:
    print("funnet")
else:
    print("ikke")
```

Maks poeng: 1

6 Oppgave 1f (2p)

Hva skrives ut når følgende kode kjøres?

```
bok = {15:5, 2:8, 20:2}
total = 0
for nokkel in bok:
    if nokkel>10:
        bok[nokkel] = bok[nokkel] * (-1)
for nokkel in bok:
    total += bok[nokkel]
print(total)
```

[ ].

Maks poeng: 2

7 Oppgave 2a (2p)

Hva skrives ut når følgende kode kjøres?

```
def inkr(a):
    a += 1
    return a

x = 1
y = inkr(x)
print(x + y)
```

[ ]

Maks poeng: 2

8 Oppgave 2b (2p)

Hva skrives ut når følgende kode kjøres?

```
def f(x):
    if x < 0:
        return -x
    if x == 0:
        return 10
    return x + 1
print(f(3) + f(0) + f(-2))
```

Maks poeng: 2

9 Oppgave 2c (3p)

#Hva skrives ut når følgende kode kjøres?

```
class A:
    def __init__(self, tall):
        self._tall = 5
        if tall > self._tall:
            self._tall = self._tall*2
    def hent(self):
        return self._tall

class B:
    def __init__(self):
        self._bok = {}
    def legg_inn(self, tall):
        self._bok[tall] = A(tall)

    def hent(self):
        total = 0
        for nokkel in self._bok:
            total = total + self._bok[nokkel].hent()
        return total
b = B()
b.legg_inn(2)
b.legg_inn(20)
print(b.hent())
```

Maks poeng: 3

10 Oppgave 3a (5p)

Fergeselskap Flyt selger billetter til biler så lenge de er maksimalt seks meter lange og maksimalt tre meter brede. Skriv funksjonen gyldig_bil som har to parametre lengde og bredde (begge flyttall), og som returnerer True dersom lengde er mindre enn 6 og bredde er mindre enn 3, ellers False. En bil med lengde nøyaktig lik 6 eller bredde nøyaktig lik 3 regnes som ugyldig.

Kallet gyldig_bil(5.5, 2.9) skal altså returnere True, mens kallet gyldig_bil(6.0, 2.0) skal returnere False, og kallet gyldig_bil(5.0, 3.5) skal returnere False.

Skriv svaret ditt her. Endringer blir lagret automatisk.

Maks poeng: 5

11 Oppgave 3b (5p)

Fergeselskapet Flyt ønsker nå å sjekke om alle bilene på kaien møter kravene. Skriv en funksjon alle_gyldige som har en parameter biler som er en nøstet liste, hvor hver indre liste består av to flyttall (lengde og bredde).

Funksjonen skal returnere True dersom alle bilene i lista er gyldige i henhold til Oppgave 3a (kall funksjonen gyldig_bil fra oppgave 3a som del av løsningen), ellers False. En tom liste som argument regnes som at alle biler er gyldige.

Kallet alle_gyldige([[5.0, 2.5], [4.8, 2.0]]) skal altså returnere True, mens kallet alle_gyldige([[5.0, 2.5], [6.1, 2.0]]) skal returnere False.

Skriv svaret ditt her. Endringer blir lagret automatisk.

Maks poeng: 5

12 Oppgave 3c (6p)

Fergeselskapet Flyt har ombestemt seg for hvordan de ønsker å representere biler i sitt program. I stedet for å ta inn en nøstet liste som argument (som i 3b) skal du nå skrive en funksjon ny_gyldige_versjon som tar inn to ulike lister som argument - en liste med lengden for hver bil, og en annen liste med bredden for de samme bilene. Lengde og bredde for en gitt bil ligger på korresponderende (samme) indeks i disse to listene.

Funksjonen skal ut fra dette lage en nøstet liste som er kompatibel med det oppgave 3b forventer å få inn, og så kalle funksjonen alle_gyldige med egnet argument. Du skal altså bare skrive kode for å endre hvordan bilene representeres, og så gjøre et kall på funksjonen fra oppgave 3b for å finne ut om alle bilene er gyldige. Du kan gjøre kall på funksjonen alle_gyldige fra oppgave 3b selv om du ikke har løst den - bare anta at funksjonen alle_gyldige gjør det den er ment å gjøre, og kall den på riktig måte fra denne nye funksjonen ny_gyldige_versjon.

Kallet ny_gyldige_versjon([5.0, 4.8], [2.5, 2.0]) skal altså returnere True av samme grunn som i oppgave 3b, mens kallet ny_gyldige_versjon([5.0, 6.1], [2.5, 2.0]) skal returnere False.

Skriv svaret ditt her. Endringer blir lagret automatisk.

Maks poeng: 6

13 Oppgave 3d (7p)

Fergeselskapet Flyt ønsker videre å sjekke om alle gyldige biler på kaien får plass på ferga. I denne deloppgava bryr man seg ikke om alle bilene er gyldige - bare om de bilene som er gyldige får plass (de ugyldige skal uansett stå igjen). Ferga har ett bildekk med en oppgitt total lengde, og vi antar for enkelhets skyld at bilene kan parkeres rett bak hverandre uten avstand.

Skriv en funksjon sjekk_lengder som har tre parametre: en liste med lengden for hver bil, og en annen liste med bredden for de samme bilene. Lengde og bredde for en gitt bil ligger på korresponderende (samme) indeks i disse to listene på samme måte som i oppgave 3c. I tillegg har funksjonen en tredje parameter total_lengde som oppgir den totale lengden på bildekket.

For hver bil skal du bruke funksjonen gyldig_bil fra oppgave 3a til å sjekke om den er gyldig. Du skal sjekke om summen av alle lengdene for gyldige biler er mindre enn eller lik total_lengde. Dersom alle gyldige biler får plass skal funksjonen returnere True, ellers False.

Kallet sjekk_lengder([4, 3, 5], [2.1, 1.9, 3.4] , 10) skal altså returnere True fordi kun de to første bilene er gyldige og 4+3 er 7, noe som er mindre enn 10. Kallet sjekk_lengder([4, 3, 5], [2.1, 1.9, 2.4] , 10) skal returnere False fordi alle de tre bilene er gyldige og 4+3+5 er 12, noe som er større enn 10.

Skriv svaret ditt her. Endringer blir lagret automatisk.

Maks poeng: 7

14 Oppgave 4a (10 poeng)

Oppgaveteksten til hele oppgave 4 ligger i et PDF dokument vedlagt svarfeltene for 4a-d. Om du synes noe er uklart i oppgaven, gjør dine egne antakelser og beskriv dem i en kommentar. Om du står fast på en deloppgave, fortsett på de som kommer etter. Du kan bruke klasser og metoder fra tidligere deloppgaver selv om du ikke har programmert dem.

Skriv svaret ditt på oppgave 4a) her (klassene Person og Bilde). Endringer blir lagret automatisk.

Maks poeng: 10

15 Oppgave 4b (14 poeng)

Oppgaveteksten til hele oppgave 4 ligger i et PDF dokument vedlagt svarfeltene for 4a-d. Om du synes noe er uklart i oppgaven, gjør dine egne antakelser og beskriv dem i en kommentar. Om du står fast på en deloppgave, fortsett på de som kommer etter. Du kan bruke klasser og metoder fra tidligere deloppgaver selv om du ikke har programmert dem.

Skriv svaret ditt på oppgave 4b) her (klassen Vegg). Endringer blir lagret automatisk.

Maks poeng: 14

16 Oppgave 4c (14 poeng)

Oppgaveteksten til hele oppgave 4 ligger i et PDF dokument vedlagt svarfeltene for 4a-d. Om du synes noe er uklart i oppgaven, gjør dine egne antakelser og beskriv dem i en kommentar. Om du står fast på en deloppgave, fortsett på de som kommer etter. Du kan bruke klasser og metoder fra tidligere deloppgaver selv om du ikke har programmert dem.

Skriv svaret ditt på oppgave 4c) her (starten på klassen Bildeplassering). Endringer blir lagret automatisk.

Maks poeng: 14

17 Oppgave 4d (12 poeng)

Oppgaveteksten til hele oppgave 4 ligger i et PDF dokument vedlagt svarfeltene for 4a-d. Om du synes noe er uklart i oppgaven, gjør dine egne antakelser og beskriv dem i en kommentar. Om du står fast på en deloppgave, fortsett på de som kommer etter. Du kan bruke klasser og metoder fra tidligere deloppgaver selv om du ikke har programmert dem.

Skriv svaret ditt på oppgave 4d) her (resten av klassen Bildeplassering). Endringer blir lagret automatisk.

Maks poeng: 12

18 Oppgave 5a (7p)

Fergeselskapet Flyt har nå kjøpt en ny ferge hvor bildekket har to filer - en fil for smale biler (biler med inntil 2 meter bredde) og en for litt bredere biler (biler med inntil 3 meter bredde). De ønsker som før å sjekke om alle bilene på kaien får plass på ferga.

Skriv en funksjon kan_stues som tar inn tre argumenter: en liste med lengden for hver bil, en annen liste med bredden for de samme bilene (på samme måte som i oppgave 3c og 3d), samt en total tilgjengelig lengde for hver av de to filene (den smale og den brede fila er like lange). Man antar nå at alle bilene er gyldige, og fokuserer bare på om de får plass på ferga.

Bilene skal stues inn på ferga på følgende måte:

Bilene kjører inn på ferga i samme rekkefølge som de ligger i listene som er argumenter til funksjonen. Så lenge det er plass, plasseres biler med bredde under 2 meter i den smale fila, og biler med bredde 2.0 meter eller over plasseres i bred fil (vi har antatt at alle biler er under 3 meter brede). Unntaket for dette er at dersom den smale fila blir full (at en bil ikke kan få plass i den smale fila uten at summen av lengdene for bilene der blir mer enn total_lengde), så kan biler med bredde under 2 meter plasseres i bred fil. Motsatt er ikke tillatt - biler som er 2 meter bred eller bredere kan ikke plasseres i smal fil. Funksjonen skal returnere True dersom alle bilene får plass i henhold til disse reglene (at summen av lengder for biler plassert i smal fil er mindre enn total_lengde og at summen av lengder for biler plassert i bred fil er mindre enn total_lengde). Ellers skal funksjonen returnere False.

Kallet kan_stues([5, 3, 5] , [1.8, 1.9, 1.4], 9) skal altså returnere True fordi de to første bilene kan plasseres i smal fil (med sum av lengde 8), mens den tredje kan legges i bred fil selv om den egentlig er smal (hvor sum lengde blir 5). Kallet kan_stues([4, 6, 4], [1.9, 1.7, 2.5], 9) skal returnere False fordi den første bilen legges i smal fil, den andre bilen da må legges i bred fil fordi det ikke er plass til den i smal fil, og hvor det da ikke er plass i bred fil til den tredje bilen (fordi total lengde i bred fil da blir 10).

Skriv svaret ditt her. Endringer blir lagret automatisk.

Maks poeng: 7

19 Oppgave 5b (5p)

Fergeselskapet Flyt ønsker nå å håndtere situasjoner hvor ikke alle bilene på kaien får plass. De ønsker å gjøre dette i henhold til ansiennitet - hvor lenge hver bil på kaien har vært medlem i fergeklubben. Skriv en funksjon ferge_plass_med_ansiennitet som har tre parametrene lengde, bredde og ansiennitet. Lengde og bredde er lister på samme måte som i oppgave 5a. Ansiennitet er en tilsvarende type liste, altså hvor ansiennitet for en gitt bil ligger på korresponderende (samme) indeks som i de to andre listene.

Plasser bilene etter de samme reglene som oppgitt i oppgave 5a - gjør dette gjennom å kalle funksjonen kan_stues fra oppgave 5a. Du kan nå anta at filene på ferga alltid er 9 meter lange. Dersom ikke alle får plass, skal funksjonen fjerne én bil av gangen inntil de resterende bilene får plass på ferga, hvor den skal starte med å fjerne bilen som har lavest ansienitet, deretter fjerne bilen med nest lavest ansienitet osv. Vi antar at alle bilene har ulik verdi for ansienitet. Funksjonen skal returnere hvor mange biler som måtte fjernes i henhold til disse reglene for at de gjenværende bilene skulle få plass. Når en bil fjernes skal gjenværende biler alltid stå i samme interne rekkefølge som de gjorde initielt (dette er av betydning grunnet hvordan biler stues).

Kallet ferge_plass_med_ansienitet([5,3,5], [1.8,1.9,1.4], [1,2,3])) skal returnere 0, fordi alle disse får plass. Kallet ferge_plass_med_ansienitet([4,6,4,3],[1.9,1.7,2.5,1.8],[2,4,3,1]) skal returnere 2, fordi den fjerde bilen har lavest ansienitet og fjernes først, uten at det er tilstrekkelig, og at også den første bilen må fjernes før de to gjenværende bilene da får plass (før et kall på kan_stues returnerer True).

Skriv svaret ditt her. Endringer blir lagret automatisk.

Maks poeng: 5

Question 14
Attached



Oppgave 4: Bildeplassering (50 poeng)

Harry har arvet et stort hus og en mengde innrammede bilder. Nå ønsker han å fylle veggene i huset med bilder, og han vil ha din hjelp til å skrive et objektorientert program i Python som kan foreslå hvordan bildene kan fordeles.

Som forberedelse har han laget en liste over hvor mange bilder det er plass til på hver vegg (heldigvis er alle bildene like store). Videre har han en liste over alle personer han har bilde av (ingen av dem har samme navn), og en liste med alle bilder som kan henges opp og hvilke personer som er med på hvert bilde.

Bilderammene er svært ulike, og Harry ønsker derfor å fordele bilder slik at hver vegg har bilder med samme type ramme. Bilder uten (synlig) ramme synes han passer sammen med alle andre og de kan derfor henges hvor som helst.

Det er noen personer det er svært viktig for Harry å få plass til, så han vil kunne velge en person av gangen og deretter plassere bilder med denne personen.

Det endelige programmet vil kunne lese all informasjon inn fra filer, finne alle bilder med en person Harry har valgt, plassere flest mulig av disse bildene på vegger slik at rammene passer sammen, og ha oversikt over hvilke bilder av personen som ikke ble plassert.

Merk: Klassene kan inneholde variabler som ikke brukes videre (i denne oppgaven). Du trenger ikke tenke på import av klasser.

4a) 10 poeng

Skriv en klasse **Person** med instansvariabler navn og relasjon – begge er strenger som får verdi fra parametere til konstruktøren. I tillegg til konstruktøren skal du skrive metodene:

- **hent_navn**
- **__str__**

Skriv også en klasse **Bilde** med instansvariabler for rammefarge, personer som er med på bildet og om bildet er hengt opp eller ikke. Konstruktøren har to parametere: Rammetype (en streng, for eksempel *"trehvit"*, *"brun"* eller *""* (en tom streng for bilder som ikke har synlig ramme) og personer på bildet (en liste med referanser til Person-objekter). I tillegg til konstruktøren skal du skrive metodene

- **hent_rammetype** som returnerer typen ramme (en streng)
- **er_med** som sjekker om en person med et oppgitt navn (parameter, en streng) er med på bildet og returnerer en boolsk verdi
- **opphengt** som returnerer en boolsk verdi som angir om bildet er hengt opp eller ikke
- **heng_opp** som kalles når bildet henges opp.

4b) 14 poeng

Skriv en klasse **Vegg** med to instansvariabler: Én som angir totalt antall plasser på veggen (parameter til konstruktøren), og én som refererer til Bilde-objekter som henger der (liste som er tom ved opprettelse). I tillegg til konstruktøren skal du skrive metodene:

- **ledig_plass** som returnerer en boolsk verdi som angir om det er plass til flere bilder
- **sjekk_ramme** som returnerer en boolsk verdi som angir om en rammetype (parameter, streng) passer med de andre bildene på veggen. Krav: Alle bilder på samme vegg må ha samme rammetype, eller usynlig ramme (""). Eks: Bilder med brun ramme kan henge sammen med bilder uten ramme, men ikke med bilder som har trehvit ramme.
- **plasser_bilder** som får en liste med referanser til Bilde-objekter i en parameter og plasserer så mange av dem det er plass til på veggen. Du kan anta at bildene har rammer som passer på veggen. Metoden oppdaterer instansvariablene for veggen og returnerer en ny liste med bilder som *ikke* fikk plass (kan være tom).

Reflekter *kort* (i svarfeltet for 4b) over mulige konsekvenser av å la en metode endre en liste som mottas i en parameter.

4c) 14 poeng

Skriv en klasse **Bildeplassering** med tre instansvariabler som representerer alle personer, bilder og vegger i systemet. Hver instansvariabel er en liste som kan holde referanser til henholdsvis personer, bilder, og vegger. Når et Bildeplassering-objekt er opprettet, er alle listene tomme. Foruten konstruktør skal du skrive følgende metoder i klassen:

- **les_personer_fra_fil** som tar et filnavn som parameter, leser filen på formatet vist nedenfor, oppretter Person-objekter og tar vare på referanser til disse. Filformat:

  navn;relasjon
  navn;relasjon

  (Klassen inneholder også metoder for å lese data om bilder og vegger fra fil, disse metodene skal du *ikke* skrive.)

- **finn_bilder** som finner alle bilder som ennå ikke er hengt opp der en person med et gitt navn (streng, parameter) er med, og returnerer disse som en liste med bilde-referanser.
- **finn_vegg** som leter gjennom veggene og returnerer en referanse til den første veggen som har ledig plass og passer for en gitt rammetype (streng, parameter til metoden). Metoden returnerer None dersom ingen vegg er funnet.

4d) 12 poeng

Utvid klassen **Bildeplassering** med metodene:

- **sorter_etter_rammetype** som tar en liste med bilde-referanser som parameter, og returnerer en ordbok der nøkler er rammetype og verdi er en liste med bilder som har rammer av den typen.
- **plasser_bilder** som tar en ordbok (nøkkel er rammetype, verdi er en liste med bilder som har den rammetypen) som parameter. Metoden plasserer bilder med en og en rammetype på vegger der de passer og det er plass, og returnerer en ny ordbok (rammetype er nøkkel, en liste med referanser til bilder er verdi) med bilder som ikke ble plassert. Det er ikke noe krav at veggene fylles maksimalt opp, bare løsningen bruker datastrukturen og metodene dine.

Question 15
Attached



Oppgave 4: Bildeplassering (50 poeng)

Harry har arvet et stort hus og en mengde innrammede bilder. Nå ønsker han å fylle veggene i huset med bilder, og han vil ha din hjelp til å skrive et objektorientert program i Python som kan foreslå hvordan bildene kan fordeles.

Som forberedelse har han laget en liste over hvor mange bilder det er plass til på hver vegg (heldigvis er alle bildene like store). Videre har han en liste over alle personer han har bilde av (ingen av dem har samme navn), og en liste med alle bilder som kan henges opp og hvilke personer som er med på hvert bilde.

Bilderammene er svært ulike, og Harry ønsker derfor å fordele bilder slik at hver vegg har bilder med samme type ramme. Bilder uten (synlig) ramme synes han passer sammen med alle andre og de kan derfor henges hvor som helst.

Det er noen personer det er svært viktig for Harry å få plass til, så han vil kunne velge en person av gangen og deretter plassere bilder med denne personen.

Det endelige programmet vil kunne lese all informasjon inn fra filer, finne alle bilder med en person Harry har valgt, plassere flest mulig av disse bildene på vegger slik at rammene passer sammen, og ha oversikt over hvilke bilder av personen som ikke ble plassert.

Merk: Klassene kan inneholde variabler som ikke brukes videre (i denne oppgaven). Du trenger ikke tenke på import av klasser.

4a) 10 poeng

Skriv en klasse **Person** med instansvariabler navn og relasjon – begge er strenger som får verdi fra parametere til konstruktøren. I tillegg til konstruktøren skal du skrive metodene:

- **hent_navn**
- **__str__**

Skriv også en klasse **Bilde** med instansvariabler for rammefarge, personer som er med på bildet og om bildet er hengt opp eller ikke. Konstruktøren har to parametere: Rammetype (en streng, for eksempel *"trehvit"*, *"brun"* eller *""* (en tom streng for bilder som ikke har synlig ramme) og personer på bildet (en liste med referanser til Person-objekter). I tillegg til konstruktøren skal du skrive metodene

- **hent_rammetype** som returnerer typen ramme (en streng)
- **er_med** som sjekker om en person med et oppgitt navn (parameter, en streng) er med på bildet og returnerer en boolsk verdi
- **opphengt** som returnerer en boolsk verdi som angir om bildet er hengt opp eller ikke
- **heng_opp** som kalles når bildet henges opp.

4b) 14 poeng

Skriv en klasse **Vegg** med to instansvariabler: Én som angir totalt antall plasser på veggen (parameter til konstruktøren), og én som refererer til Bilde-objekter som henger der (liste som er tom ved opprettelse). I tillegg til konstruktøren skal du skrive metodene:

- **ledig_plass** som returnerer en boolsk verdi som angir om det er plass til flere bilder
- **sjekk_ramme** som returnerer en boolsk verdi som angir om en rammetype (parameter, streng) passer med de andre bildene på veggen. Krav: Alle bilder på samme vegg må ha samme rammetype, eller usynlig ramme (""). Eks: Bilder med brun ramme kan henge sammen med bilder uten ramme, men ikke med bilder som har trehvit ramme.
- **plasser_bilder** som får en liste med referanser til Bilde-objekter i en parameter og plasserer så mange av dem det er plass til på veggen. Du kan anta at bildene har rammer som passer på veggen. Metoden oppdaterer instansvariablene for veggen og returnerer en ny liste med bilder som *ikke* fikk plass (kan være tom).

Reflekter *kort* (i svarfeltet for 4b) over mulige konsekvenser av å la en metode endre en liste som mottas i en parameter.

4c) 14 poeng

Skriv en klasse **Bildeplassering** med tre instansvariabler som representerer alle personer, bilder og vegger i systemet. Hver instansvariabel er en liste som kan holde referanser til henholdsvis personer, bilder, og vegger. Når et Bildeplassering-objekt er opprettet, er alle listene tomme. Foruten konstruktør skal du skrive følgende metoder i klassen:

- **les_personer_fra_fil** som tar et filnavn som parameter, leser filen på formatet vist nedenfor, oppretter Person-objekter og tar vare på referanser til disse. Filformat:

  navn;relasjon
  navn;relasjon

  (Klassen inneholder også metoder for å lese data om bilder og vegger fra fil, disse metodene skal du *ikke* skrive.)

- **finn_bilder** som finner alle bilder som ennå ikke er hengt opp der en person med et gitt navn (streng, parameter) er med, og returnerer disse som en liste med bilde-referanser.
- **finn_vegg** som leter gjennom veggene og returnerer en referanse til den første veggen som har ledig plass og passer for en gitt rammetype (streng, parameter til metoden). Metoden returnerer None dersom ingen vegg er funnet.

4d) 12 poeng

Utvid klassen **Bildeplassering** med metodene:

- **sorter_etter_rammetype** som tar en liste med bilde-referanser som parameter, og returnerer en ordbok der nøkler er rammetype og verdi er en liste med bilder som har rammer av den typen.
- **plasser_bilder** som tar en ordbok (nøkkel er rammetype, verdi er en liste med bilder som har den rammetypen) som parameter. Metoden plasserer bilder med en og en rammetype på vegger der de passer og det er plass, og returnerer en ny ordbok (rammetype er nøkkel, en liste med referanser til bilder er verdi) med bilder som ikke ble plassert. Det er ikke noe krav at veggene fylles maksimalt opp, bare løsningen bruker datastrukturen og metodene dine.

Question 16
Attached



Oppgave 4: Bildeplassering (50 poeng)

Harry har arvet et stort hus og en mengde innrammede bilder. Nå ønsker han å fylle veggene i huset med bilder, og han vil ha din hjelp til å skrive et objektorientert program i Python som kan foreslå hvordan bildene kan fordeles.

Som forberedelse har han laget en liste over hvor mange bilder det er plass til på hver vegg (heldigvis er alle bildene like store). Videre har han en liste over alle personer han har bilde av (ingen av dem har samme navn), og en liste med alle bilder som kan henges opp og hvilke personer som er med på hvert bilde.

Bilderammene er svært ulike, og Harry ønsker derfor å fordele bilder slik at hver vegg har bilder med samme type ramme. Bilder uten (synlig) ramme synes han passer sammen med alle andre og de kan derfor henges hvor som helst.

Det er noen personer det er svært viktig for Harry å få plass til, så han vil kunne velge en person av gangen og deretter plassere bilder med denne personen.

Det endelige programmet vil kunne lese all informasjon inn fra filer, finne alle bilder med en person Harry har valgt, plassere flest mulig av disse bildene på vegger slik at rammene passer sammen, og ha oversikt over hvilke bilder av personen som ikke ble plassert.

Merk: Klassene kan inneholde variabler som ikke brukes videre (i denne oppgaven). Du trenger ikke tenke på import av klasser.

4a) 10 poeng

Skriv en klasse **Person** med instansvariabler navn og relasjon – begge er strenger som får verdi fra parametere til konstruktøren. I tillegg til konstruktøren skal du skrive metodene:

- **hent_navn**
- **__str__**

Skriv også en klasse **Bilde** med instansvariabler for rammefarge, personer som er med på bildet og om bildet er hengt opp eller ikke. Konstruktøren har to parametere: Rammetype (en streng, for eksempel *"trehvit"*, *"brun"* eller *""* (en tom streng for bilder som ikke har synlig ramme) og personer på bildet (en liste med referanser til Person-objekter). I tillegg til konstruktøren skal du skrive metodene

- **hent_rammetype** som returnerer typen ramme (en streng)
- **er_med** som sjekker om en person med et oppgitt navn (parameter, en streng) er med på bildet og returnerer en boolsk verdi
- **opphengt** som returnerer en boolsk verdi som angir om bildet er hengt opp eller ikke
- **heng_opp** som kalles når bildet henges opp.

4b) 14 poeng

Skriv en klasse **Vegg** med to instansvariabler: Én som angir totalt antall plasser på veggen (parameter til konstruktøren), og én som refererer til Bilde-objekter som henger der (liste som er tom ved opprettelse). I tillegg til konstruktøren skal du skrive metodene:

- **ledig_plass** som returnerer en boolsk verdi som angir om det er plass til flere bilder
- **sjekk_ramme** som returnerer en boolsk verdi som angir om en rammetype (parameter, streng) passer med de andre bildene på veggen. Krav: Alle bilder på samme vegg må ha samme rammetype, eller usynlig ramme (""). Eks: Bilder med brun ramme kan henge sammen med bilder uten ramme, men ikke med bilder som har trehvit ramme.
- **plasser_bilder** som får en liste med referanser til Bilde-objekter i en parameter og plasserer så mange av dem det er plass til på veggen. Du kan anta at bildene har rammer som passer på veggen. Metoden oppdaterer instansvariablene for veggen og returnerer en ny liste med bilder som *ikke* fikk plass (kan være tom).

Reflekter *kort* (i svarfeltet for 4b) over mulige konsekvenser av å la en metode endre en liste som mottas i en parameter.

4c) 14 poeng

Skriv en klasse **Bildeplassering** med tre instansvariabler som representerer alle personer, bilder og vegger i systemet. Hver instansvariabel er en liste som kan holde referanser til henholdsvis personer, bilder, og vegger. Når et Bildeplassering-objekt er opprettet, er alle listene tomme. Foruten konstruktør skal du skrive følgende metoder i klassen:

- **les_personer_fra_fil** som tar et filnavn som parameter, leser filen på formatet vist nedenfor, oppretter Person-objekter og tar vare på referanser til disse. Filformat:

  navn;relasjon
  navn;relasjon

  (Klassen inneholder også metoder for å lese data om bilder og vegger fra fil, disse metodene skal du *ikke* skrive.)

- **finn_bilder** som finner alle bilder som ennå ikke er hengt opp der en person med et gitt navn (streng, parameter) er med, og returnerer disse som en liste med bilde-referanser.
- **finn_vegg** som leter gjennom veggene og returnerer en referanse til den første veggen som har ledig plass og passer for en gitt rammetype (streng, parameter til metoden). Metoden returnerer None dersom ingen vegg er funnet.

4d) 12 poeng

Utvid klassen **Bildeplassering** med metodene:

- **sorter_etter_rammetype** som tar en liste med bilde-referanser som parameter, og returnerer en ordbok der nøkler er rammetype og verdi er en liste med bilder som har rammer av den typen.
- **plasser_bilder** som tar en ordbok (nøkkel er rammetype, verdi er en liste med bilder som har den rammetypen) som parameter. Metoden plasserer bilder med en og en rammetype på vegger der de passer og det er plass, og returnerer en ny ordbok (rammetype er nøkkel, en liste med referanser til bilder er verdi) med bilder som ikke ble plassert. Det er ikke noe krav at veggene fylles maksimalt opp, bare løsningen bruker datastrukturen og metodene dine.

Question 17
Attached



Oppgave 4: Bildeplassering (50 poeng)

Harry har arvet et stort hus og en mengde innrammede bilder. Nå ønsker han å fylle veggene i huset med bilder, og han vil ha din hjelp til å skrive et objektorientert program i Python som kan foreslå hvordan bildene kan fordeles.

Som forberedelse har han laget en liste over hvor mange bilder det er plass til på hver vegg (heldigvis er alle bildene like store). Videre har han en liste over alle personer han har bilde av (ingen av dem har samme navn), og en liste med alle bilder som kan henges opp og hvilke personer som er med på hvert bilde.

Bilderammene er svært ulike, og Harry ønsker derfor å fordele bilder slik at hver vegg har bilder med samme type ramme. Bilder uten (synlig) ramme synes han passer sammen med alle andre og de kan derfor henges hvor som helst.

Det er noen personer det er svært viktig for Harry å få plass til, så han vil kunne velge en person av gangen og deretter plassere bilder med denne personen.

Det endelige programmet vil kunne lese all informasjon inn fra filer, finne alle bilder med en person Harry har valgt, plassere flest mulig av disse bildene på vegger slik at rammene passer sammen, og ha oversikt over hvilke bilder av personen som ikke ble plassert.

Merk: Klassene kan inneholde variabler som ikke brukes videre (i denne oppgaven). Du trenger ikke tenke på import av klasser.

4a) 10 poeng

Skriv en klasse **Person** med instansvariabler navn og relasjon – begge er strenger som får verdi fra parametere til konstruktøren. I tillegg til konstruktøren skal du skrive metodene:

- **hent_navn**
- **__str__**

Skriv også en klasse **Bilde** med instansvariabler for rammefarge, personer som er med på bildet og om bildet er hengt opp eller ikke. Konstruktøren har to parametere: Rammetype (en streng, for eksempel *"trehvit"*, *"brun"* eller *""* (en tom streng for bilder som ikke har synlig ramme) og personer på bildet (en liste med referanser til Person-objekter). I tillegg til konstruktøren skal du skrive metodene

- **hent_rammetype** som returnerer typen ramme (en streng)
- **er_med** som sjekker om en person med et oppgitt navn (parameter, en streng) er med på bildet og returnerer en boolsk verdi
- **opphengt** som returnerer en boolsk verdi som angir om bildet er hengt opp eller ikke
- **heng_opp** som kalles når bildet henges opp.

4b) 14 poeng

Skriv en klasse **Vegg** med to instansvariabler: Én som angir totalt antall plasser på veggen (parameter til konstruktøren), og én som refererer til Bilde-objekter som henger der (liste som er tom ved opprettelse). I tillegg til konstruktøren skal du skrive metodene:

- **ledig_plass** som returnerer en boolsk verdi som angir om det er plass til flere bilder
- **sjekk_ramme** som returnerer en boolsk verdi som angir om en rammetype (parameter, streng) passer med de andre bildene på veggen. Krav: Alle bilder på samme vegg må ha samme rammetype, eller usynlig ramme (""). Eks: Bilder med brun ramme kan henge sammen med bilder uten ramme, men ikke med bilder som har trehvit ramme.
- **plasser_bilder** som får en liste med referanser til Bilde-objekter i en parameter og plasserer så mange av dem det er plass til på veggen. Du kan anta at bildene har rammer som passer på veggen. Metoden oppdaterer instansvariablene for veggen og returnerer en ny liste med bilder som *ikke* fikk plass (kan være tom).

Reflekter *kort* (i svarfeltet for 4b) over mulige konsekvenser av å la en metode endre en liste som mottas i en parameter.

4c) 14 poeng

Skriv en klasse **Bildeplassering** med tre instansvariabler som representerer alle personer, bilder og vegger i systemet. Hver instansvariabel er en liste som kan holde referanser til henholdsvis personer, bilder, og vegger. Når et Bildeplassering-objekt er opprettet, er alle listene tomme. Foruten konstruktør skal du skrive følgende metoder i klassen:

- **les_personer_fra_fil** som tar et filnavn som parameter, leser filen på formatet vist nedenfor, oppretter Person-objekter og tar vare på referanser til disse. Filformat:

  navn;relasjon
  navn;relasjon

  (Klassen inneholder også metoder for å lese data om bilder og vegger fra fil, disse metodene skal du *ikke* skrive.)

- **finn_bilder** som finner alle bilder som ennå ikke er hengt opp der en person med et gitt navn (streng, parameter) er med, og returnerer disse som en liste med bilde-referanser.
- **finn_vegg** som leter gjennom veggene og returnerer en referanse til den første veggen som har ledig plass og passer for en gitt rammetype (streng, parameter til metoden). Metoden returnerer None dersom ingen vegg er funnet.

4d) 12 poeng

Utvid klassen **Bildeplassering** med metodene:

- **sorter_etter_rammetype** som tar en liste med bilde-referanser som parameter, og returnerer en ordbok der nøkler er rammetype og verdi er en liste med bilder som har rammer av den typen.
- **plasser_bilder** som tar en ordbok (nøkkel er rammetype, verdi er en liste med bilder som har den rammetypen) som parameter. Metoden plasserer bilder med en og en rammetype på vegger der de passer og det er plass, og returnerer en ny ordbok (rammetype er nøkkel, en liste med referanser til bilder er verdi) med bilder som ikke ble plassert. Det er ikke noe krav at veggene fylles maksimalt opp, bare løsningen bruker datastrukturen og metodene dine.