Informasjon

Oppgave	Tittel	Maks poeng	Oppgavetype
i	Informasjon		Informasjon eller ressurser

Oppgave 1 (13 poeng)

Oppgave	Tittel	Maks poeng	Oppgavetype
1(a)	Oppgave 1a)	1	Fyll inn tekst
1(b)	Oppgave 1b)	1	Fyll inn tekst
1(c)	Oppgave 1c)	1	Fyll inn tekst
1(d)	Oppgave 1d)	3	Fyll inn tekst
1(e)	Oppgave 1e)	1	Fyll inn tekst
1(f)	Oppgave 1f)	1	Fyll inn tekst
1(g)	Oppgave 1g)	2	Fyll inn tekst
1(h)	Oppgave 1h)	3	Fyll inn tekst

Oppgave 2 (11 poeng)

Oppgave	Tittel	Maks poeng	Oppgavetype
2(a)	Oppgave 2a)	2	Fyll inn tekst
2(b)	Oppgave 2b)	4	Fyll inn tekst
2(c)	Oppgave 2c)	5	Fyll inn tekst

Oppgave 3 (30 poeng)

Oppgave	Tittel	Maks poeng	Oppgavetype

3(a)	Oppgave 3a	10	Programmering
3(b)	Oppgave 3b	5	Programmering
3(c)	Oppgave 3c	5	Programmering
3(d)	Oppgave 3d	5	Programmering
3(e)	Oppgave 3e)	5	Langsvar

Oppgave 4 (60 poeng)

Oppgave	Tittel	Maks poeng	Oppgavetype
4(a)	Oppgave 4-01)	5	Filopplasting
4(b)	Oppgave 4-02)	5	Filopplasting
4(c)	Oppgave 4a-g)	50	Programmering

Oppgave 5 (6 poeng)

Oppgave	Tittel	Maks poeng	Oppgavetype
5	Oppgave 5	6	Programmering

Informasjon

Prøveeksamen i IN1000 høsten 2020

Alle hjelpemidler er tillatt (lærebok, nettressurser, notater osv.)

Det er *ikke tillatt* å samarbeide eller kommunisere med andre under eksamen om oppgavene.

Man kan trekkes ut til samtale for å kontrollere eierskap til sin besvarelse, jf. https://www.mn.uio.no/om/hms/koronavirus/kontrollsamtale/. Samtalen har ikke innvirkning på sensuren/karakteren, men kan lede til at instituttet oppretter fuskesak. Les mer om hva som regnes som fusk på UiOs nettsider: https://www.uio.no/om/regelverk/studier/studier-eksamener/fuskesaker/

Dette oppgavesettet inneholder oppgaver som gir totalt 120 poeng. Eksamen vil inneholde oppgaver som gir totalt 100 poeng.

1(a) Oppgave 1a)

```
tall1 = 1+2+3
tall2 = 2
tall1 =
```

Skriv et uttrykk i svarfeltet på høyresiden av siste tilordning som gir **tall1** verdien **11**. Uttrykket skal inkludere **tall1**, **tall2** og tallet **1,** men kun én gang hver**.**

Maks poeng: 1

1(b) Oppgave 1b)

```
tall = 4*2
t1 = "a"
if tall>9:
    t1 = t1 + "a"
print(          )
```

Fyll ut det som mangler for at denne koden skal printe ut strengen **ba** .
Det du fyller ut skal inkludere bruk av variabelen **t1**.

Maks poeng: 1

1(c) Oppgave 1c)

Fyll ut en logisk (boolsk) operator i svarfeltet som gjør at denne koden skriver ut strengen **ja** .

```
tall1 = 2*3
tall2 = 2**3
if (tall1>7)          (tall2>7):
    print("ja")
else:
    print("nei")
```

Maks poeng: 1

1(d) Oppgave 1d)

Hvilke verdier må ligge i variabelen **liste** ved starten av denne koden for at hele koden skal kjøre (for at **liste** skal ha lengde 3 og for at **tall** skal ha verdien 12 i siste linje)?

Verdiene i liste skal være positive heltall større enn 0.

Skriv svaret som man skriver en python-liste, f.eks. **[1,1,1]** men med verdier i **liste** som gjør at koden kjører riktig.

```
#liste = ??
tall = 0
for i in liste:
    tall+=1
    tall=tall*i
assert len(liste)==3
assert tall==12
```

Maks poeng: 3

1(e) Oppgave 1e)

Fyll inn tallet som mangler for at tallet **30** skal skrives ut:

```
a=3
for i in range (0, [          ] ) :
   a = a*2
   for b in [1,2,3]:
      a += b
print(a)
```

Maks poeng: 1

1(f) Oppgave 1f)

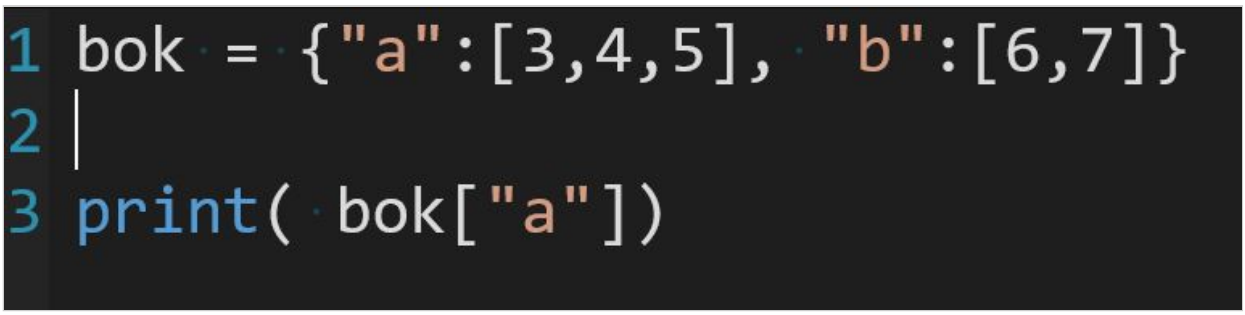
Noe mangler i uttrykket i linje 7 for at koden vil skrive ut tallet **2** til terminalen. Skriv det komplette uttrykket i svarfeltet.

```
tall = 0
operasjoner = "idd.d..did"
for operasjon in operasjoner:
    if operasjon == "d":
        tall = tall*2
    elif operasjon == "i":
        tall = tall                 # linje 7
    else:
        tall = 0
print(tall)
```

Skriv ditt svar her:

Maks poeng: 1

1(g) Oppgave 1g)



Det mangler en programsetning i linje 2 i vedlagte kode for at følgende skrives ut på terminalen:
[3, 4, 5, 8]

Programsetningen skal inneholde et metodekall på **bok["a"]** .
Det skal altså ikke skal gjøres en tilordning til **bok["a"]** .

Skriv hele setningen som mangler i linje 2:

Maks poeng: 2

1(h) Oppgave 1h)

```
stater = {"AL":"Alabama", "AK":"Alaska", "AZ":"Arizona", "AR":"Arkansas", "CA":"California", \
        "CO":"Colorado", "CT":"Connecticut", "DE":"Delaware", "FL":"Florida", "GA":"Georgia", \
        "HI":"Hawaii", "ID":"Idaho", "IL":"Illinois", "IN":"Indiana", \
        "IA":"Iowa", "KS":"Kansas", "KY":"Kentucky", "LA":"Louisiana", \
        "ME":"Maine", "MD":"Maryland", "MA":"Massachusetts", "MI":"Michigan", \
        "MN":"Minnesota", "MS":"Mississippi", "MO":"Missouri", "MT":"Montana", \
        "NE":"Nebraska", "NV":"Nevada", "NH":"New Hampshire", "NJ":"New Jersey", \
        "NM":"New Mexico", "NY":"New York", "NC":"North Carolina", "ND":"North Dakota", \
        "OH":"Ohio", "OK":"Oklahoma", "OR":"Oregon", "PA":"Pennsylvania", \
        "RI":"Rhode Island", "SC":"South Carolina", "SD":"South Dakota", "TN":"Tennessee", \
        "TX":"Texas", "UT":"Utah", "VT":"Vermont", "VA":"Virginia", \
        "WA":"Washington", "WV":"West Virginia", "WI":"Wisconsin", "WY":"Wyoming", \
        "AS":"American Samoa", "DC":"District of Columbia", "FM":"Federated States of
Micronesia", "GU":"Guam", \
        "MH":"Marshall Islands", "MP":"Northern Mariana Islands", "PW":"Palau", "PR":"Puerto
Rico", \
        "VI":"Virgin Islands" \
        }

# statkoder er ei liste med nøklene i ordboka stater i den samme rekkefølgen som ovenfor.
# statkoder = ["AL", "AK", .... , "VI"]

maks = 0
ny = ""
for k in statkoder:
    navn = stater[k].split()
    if len(navn) > maks:
        maks = len(navn)
        ny = k
print(ny, stater[k])
```

Hva skrives ut her?

Maks poeng: 3

2(a) Oppgave 2a)

```
1 def funk(a, b):
2     
3
4 c = funk(2+3, 2) * 4
5 print(c)
```

Hva er det enkleste uttykket der parametrene a og b inngår, som kan returneres fra funksjonen i linje 3 for at vedlagte kode vil skrive ut tallet **40**? Ta med nøkkelordet **return**, for eksempel **return xxx**
(men erstatt **xxx** med korrekt uttrykk)

Skriv ditt svar her:

Maks poeng: 2

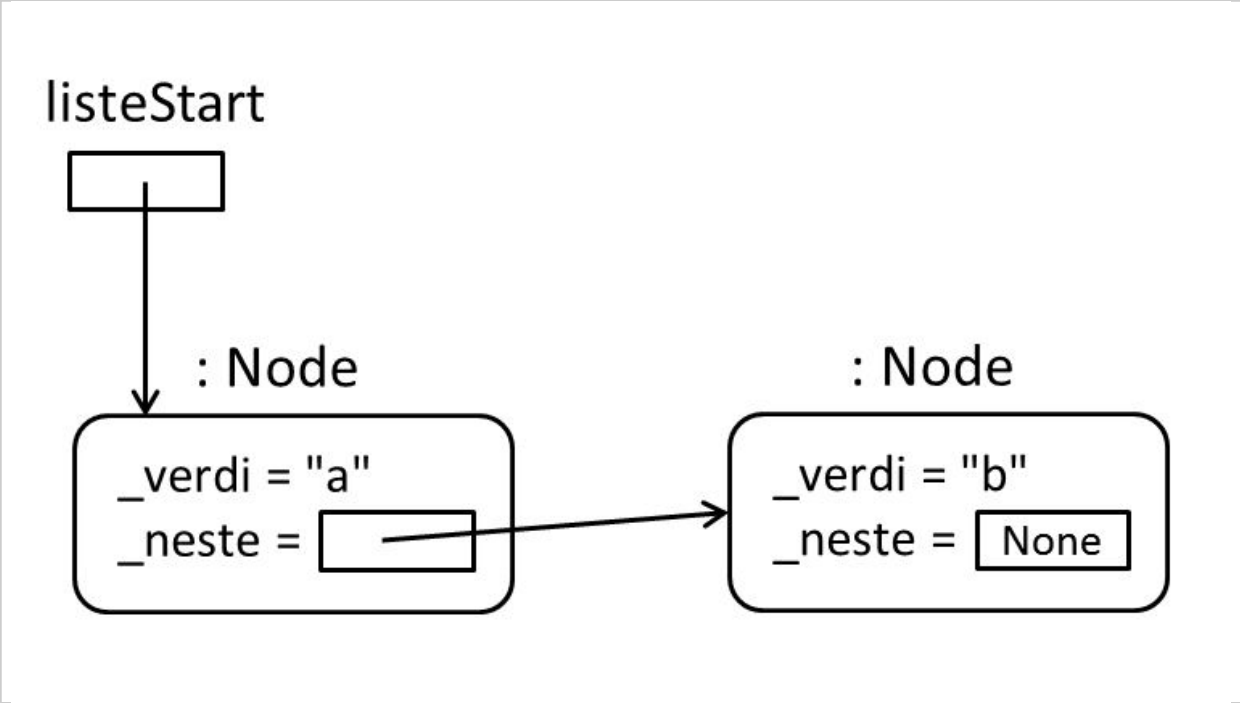
2(b) Oppgave 2b)

Hvilken ordning av føgende programsetninger vil gi et kjørende program som skriver ut 16 (og kun det)? Bare noen av programsetningene under skal brukes, og du kan bruke samme programsetning flere ganger i programmet for å oppnå ønsket funksjonalitet. Klasse- og metodedefinisjoner skal beholde rekkefølgen under.

a) class MyClass:
b) def __init__(self, value):
c) def change(self):
d) def conclude(self):
e) self._number = value
f) self._value = value
g) self._number = self._number * 2
h) return self._number
i) number = 3
j) myObj.change()
k) print(myObj.conclude())
l) myObj = MyClass(number+1)
m) myObj = MyClass(number)

Maks poeng: 4

2(c) Oppgave 2c)



Gitt klassen **Node** og et hovedprogram som vist nedenfor, oppgi programsetningen som mangler i metoden **settInn** for at et kall på **hovedprogram** vil opprette 2 objekter med verdiene "a" og "b" i en struktur som vist i figuren.

I figuren er en variabel (lokale og instansvariabler) vist som en firkantet boks, piler angir objektreferanser, og objektene er vist som rektangler med avrundede hjørner.
Variabelen **listeStart** refererer til objektet med verdi "a".

```
class Node :
    def __init__ (self, verdi) :
        self._verdi = verdi
        self._neste = None

    def settInn (self, ny) :


def hovedprogram():
    listeStart = Node("a")
    ny = Node ("b")
    listeStart.settInn(ny)
```

Skriv ditt svar her (kun den ene setningen som mangler i metoden **settinn**

Maks poeng: 5

3(a) Oppgave 3a

Knut vil beregne pris inkludert frakt når han skal handle på nettet.

Skriv en funksjon **pris_inkl_frakt(varepris)**.
Dersom varepris er over 1000 kroner legges det ikke på noe kostnad til frakt og funksjonen skal returnere samme varepris som den fikk inn som argument. Dersom varepris er mellom 500 og 1000 kroner (fra og med 500, til og med 1000) skal det legges på frakt, slik at funksjonen returnerer en pris som er 50 kroner høyere enn varepris. Dersom varepris er under 500 kroner skal det både legges på frakt og ekstragebyr, slik at funksjonen skal returnere en pris som er 80 kroner høyere enn varepris.

Med andre ord skal f.eks. kallet pris_inkl_frakt(300) evaluere til 380, pris_inkl_frakt(600) evaluere til 650, og pris_inkl_frakt(1300) evaluere til 1300.

Skriv ditt svar her

Maks poeng: 10

3(b) Oppgave 3b

Per vil lage seg en handleliste og raskt kunne fjerne utsolgte varer fra denne lista.

Skriv en funksjon **fjern_utsolgte(handleliste, utsolgte).**
Funksjonen tar som argumenter en liste med navn på varer (handleliste: liste av streng-verdier) og en liste over utsolgte varer (utsolgte: liste av streng-verdier). Funksjonen skal returnere en ny liste som inneholde de ønskede varene som ikke er utsolgte, dvs en ny liste som inneholder alle streng-verdier som finnes i listen handleliste, men ikke finnes i listen utsolgte. I tillegg skal funksjonen skrive til terminalen navnet på hver vare fra handlelisten som er utsolgt.

Altså skal for eksempel kallet fjern_utsolgte(["melk", "brus", "pasta"], ["kanel","brus"]) evaluere til listen ["melk", "pasta"], samtidig som teksten brus skrives til terminalen.

Skriv ditt svar her

Maks poeng: 5

3(c) Oppgave 3c

Lise skal reise som backpacker gjennom flere land i asia og vil finne ut hvilke vaksiner hun må ta før reisen.

Skriv en funksjon **samlet_vaksinasjon(krav_hvert_land).**
Funksjonen tar som argument en liste av lister hvor hver indre liste er hvilke vaksiner som trengs for et av landene hun skal reise til. Funksjonen skal returnere en samlet liste over hvilke vaksiner som trengs, dvs en liste av streng-verdier som inkluderer alle vaksinene funnet i et av landene. Dersom samme vaksine finnes i lista for mange ulike land, skal den bare være med én gang i den returnerte lista.

Altså skal for eksempel kallet samlet_vaksinasjon([["difteri","tyfoid"], ["hepatit","difteri"]]) evaluere til en liste ['difteri', 'tyfoid', 'hepatit'] (rekkefølgen av vaksinene i den returnerte lista er uviktig).

Skriv ditt svar her

Maks poeng: 5

3(d) Oppgave 3d

Koden under definerer en variabel setning og lager varianter av denne setningen hvor først ordet "en" er fjernet og deretter også ordet "skal" er fjernet. Det er store likheter mellom kodelinjene som lager setning uten ordet "en" og kodelinjene som lager setning uten ordet "skal". Skriv en alternativ versjon av programmet som unngår denne redundansen ved å definere en funksjon forkort_setning og som kaller denne funksjonen to ganger for å lage setning uten henholdsvis ordene "en" og "skal". Skriv det fulle resulterende programmet, inkludert definisjon av funksjonen (og dens parametre), kall på funksjonen (med argumenter) og andre kodelinjer som trengs for å oppnå samme funksjonalitet som den opprinnelige koden under.

```
setning = "en krabbe skal en dag ut av skallet "

#fjerner alle ord "en":
setning_v2 = ""
for ord in setning.split():
    if not ord=="en":
        setning_v2 = setning_v2 + ord + " "

#fjerner alle ord "skal":
setning_v3 = ""
for ord in setning_v2.split():
    if not ord=="skal":
        setning_v3 = setning_v3 + ord + " "

print(setning_v3) #krabbe dag ut av skallet
```

Skriv ditt svar her

Maks poeng: 5

3(e) Oppgave 3e)

Oppgi tre uttrykk som du finner i dine svar på oppgavene 3a) til 3d). Uttrykkene skal evaluere til verdier av hver sin type. For hvert uttrykk, oppgi hvilken deloppgave du har besvart med koden du har hentet uttrykket fra.

Du skal altså skrive tre linjer. På hver linje skriver du selve uttrykket (ikke en hel programsetning) fulgt av en parentes der du oppgir deloppgave, for eksempel **(3b)**.

Skriv ditt svar her

Maks poeng: 5

4(a) Oppgave 4-01)

For prøveeksamen er det lagt til to ekstra deloppgaver i oppgave 4, nummerert 4-01 og 4-02. Resten av oppgave 4 (4a til 4g) er identisk med fjorårets eksamen.

Les gjennom hele den vedlagte oppgave 4 (samme pdf i alle deloppgaver). Tegn deretter et UML klassediagram som viser alle klasser du er bedt om å implementere (kun klassenavn) og relasjonene mellom dem. For hver relasjon skal du ta med antall og navn (hva relasjonen representerer).

Du kan tegne diagrammet på papir, og laste opp et bilde av tegningen - eventuelt bruke et tegneprogram du kjenner fra før (ellers går det antakelig raskere å tegne for hånd og ta et bilde).

Maks poeng: 5

4(b) Oppgave 4-02)

Lag deretter en ny tegning, med et klassediagram som kun viser klassen Emne. Her skal du ta med alle instansvariabler og metoder i Emne. Angi type for instansvariabler, parametere og returverdier, og om instansvariabler og metoder er public (tilgjengelige i klassens grensesnitt) eller ikke. Det meste av dette er gitt i oppgaveteksten, noe vil du kanskje bestemme underveis i oppgave-løsingen.

Du kan tegne diagrammet på papir, og laste opp et bilde av tegningen - eventuelt bruke et tegneprogram du kjenner fra før (ellers går det antakelig raskere å tegne for hånd og ta et bilde)

Maks poeng: 5

4(c) Oppgave 4a-g)

Denne oppgaven består av en rekke deloppgaver der du skal implementere komponenter av et større system. Om du hopper over en deloppgave er det likevel viktig at du leser hele oppgaveteksten i vedlagte pdf-dokument.

Svar på deloppgave a) til g) legges samlet i dette svarfeltet, fordelt på de klassene som er spesifisert. Skriv en kommentar der du angir hvilken deloppgave du besvarer foran tilhørende kode.

Skriv all kode til oppgave 4a-g her:

Maks poeng: 50

5 Oppgave 5

Karl vil kunne sikre seg at han er forskånt for å måtte lese et bestemt fy-ord og dets synomer.

Skriv en funksjon **sjekk_om_fyord(setning, fyord, synonym_liste)**.
Funksjonen tar som argument en setning (en streng bestående av flere ord med mellomrom mellom) , et fyord (en streng bestående av ett enkelt ord uten noen mellomrom) og en samling av ulike synonymer (en liste av lister, hvor hver indre liste består av streng-verdier som er enkeltord). Funksjonen skal sjekke om minst ett av ordene i setningen er et synonym med det oppgitte fyordet (ifølge den oppgitte samlingen av synonymer) eller om fyordet i seg selv finnes i setningen. Dersom fyordet eller et synonym av fyordet finnes i setningen skal funksjonen returnere True. Hvis ikke skal funksjonen returnere False.

Altså skal for eksempel kallet *sjekk_om_fyord("spis masse godsaker", "snop", [["saft","lemonade"],["snacks","snop","godsaker"],["mye","masse"]])* returnere True, mens kallet *sjekk_om_fyord("spis masse godsaker", "lemonade", [["saft","lemonade"], ["snacks","snop","godsaker"],["mye","masse"]])* skal returnere False.

Skriv ditt svar her

Maks poeng: 6

Question 17
Attached

Oppgave 4 Administrasjon av obligatoriske oppgaver (50 poeng)

Universitetet i Ruritania skal digitalisere sin håndtering av studentenes obligatoriske oppgaver («obliger»), og du skal lage deler av en pilot med et objektorientert design basert på innkapsling, dvs. instansvariable og non-public metoder (navn starter med _) skal kun benyttes i den klassen de er definert. **Det er i mange av deloppgavene nødvendig å bruke metoder fra andre klasser/ deloppgaver. Du kan bruke disse selv om du ikke har skrevet dem.**

Emnene har ulike antall obligatoriske oppgaver som kreves godkjent, og hver oblig i et emne må registreres manuelt hvert semester. En student må ha alle oppgaver gitt i emnet godkjent for å få gå opp til eksamen, men i denne piloten foretas det ikke kontroll av om studenten har fått godkjent forrige oblig før godkjenning av besvarelse for neste oblig. Foruten obliger har et emne registrerte studenter og ansatte rettere. Både rettere og studenter har unike brukernavn.

Innleveringen av besvarelser for en oblig foregår i et annet system som du ikke skal lage. Dette systemet lager en fil for hver oblig, med et entydig navn etterfulgt av «.txt», for eksempel *oblig1.txt*. Filen inneholder en linje per student på følgende format (eksempel med to linjer):
student1 obl1student1.txt
student2

I eksempelet er *student1* et entydig brukernavn og *obl1student1.txt* er navnet på filen der student1s besvarelse ligger. *student2* i eksempelet har ikke levert noen besvarelse på oblig 1.

Hver besvarelse skal vurderes av en retter for emnet. Når fristen for en obligatorisk oppgave er utløpt, fordeles besvarelser på retterne, som vurderer en og en til godkjent (1) eller underkjent (0).

Piloten har funksjonalitet for hente ut en eksamensliste med brukernavn på alle studenter som er kvalifisert for eksamen – dvs har godkjent alle registrerte obliger i emnet.

Oppgave 4a) 5 poeng

Definer en klasse **Emne** (inkludert konstruktør) som holder orden på og administrerer obliger, rettere og studenter i et emne. Konstruktørens parametere gir verdier til emnekode (streng), registrerte studenter (ordbok med Student-objekter) og rettere (liste med Retter-objekter).

Oppgave 4b) 5 poeng

Skriv metoden **administrer** i klassen Emne. Metoden skriver ut emnekoden for emnet og en meny på terminalen før den ber om kommando. Den tar imot følgende lovlige kommandoer fra en bruker:
O: Ny oblig
F: Frist ute, start retting
L: Lag eksamensliste
A: Avslutt

Annen input skal gi feilmelding og nytt forsøk. En kommando bør gjenkjennes selv om det er blanke foran eller bak, og uansett om brukeren skriver liten eller stor bokstav (metodene **strip** og **upper** er nyttige her). For å utføre kommandoene skal metoden kalle på følgende non-public metoder som også ligger i klassen **Emne**:
O -> **_opprettOblig**
F -> **_startRetting**
L -> **_skrivEksamensListe**
Disse metodene skal skrives i senere deloppgaver. De opererer kun på instansvariabler i klassen, og har ingen parametere (annet enn **self**) eller returverdier.

Oppgave 4c) 10 poeng

Skriv en klasse **Student** med instansvariabler for brukernavn og fullt navn (begge får verdi fra parametere til konstruktøren) og en ordbok som skal ta vare på resultater av rettinger, der obligId er nøkkel og resultatet verdien. Klassen har følgende metoder i tillegg til konstruktør:

registrer med parametere **oblig** (en streng som identifiserer en oblig entydig) og **resultat** (et heltall som angir om en besvarelse er vurdert som godkjent). Metoden registrerer hvilket resultat en student har fått på en oblig.

altGodkjent med parameter **antObliger** som angir antall obliger registrert i emnet. Metoden returnerer en boolsk verdi: **True** hvis studenten har fått alle obliger godkjent, **False** om en eller flere obliger mangler eller har et annet resultat enn godkjent (kodet som heltallet 1).

Oppgave 4d) 2 poeng

Skriv klassen **Retter** med instansvariabel retterens brukernavn (parameter til konstruktøren) og metoden:

vurder med en parameter som angir et filnavn for en besvarelse. I denne foreløpige utgaven av piloten ignorerer programmet ditt filen med besvarelsen, og returnerer alltid 1 (godkjent).

Oppgave 4e) 8 poeng

Skriv en klasse **Oblig** med tre instansvariabler: Entydig id for obligen, en leveringsfrist på formen ååmmdd (f. eks. 190906 for 6. september 2019) og om obligen er rettet eller ikke. De to første initialiseres med parametere til konstruktøren. Skriv følgende metoder i klassen Oblig:

klarForRetting med parameter som angir dagens dato. Metoden returnerer **True** hvis fristen er før dagens dato og besvarelsene for obligen ikke allerede er rettet – ellers **False**.

hentBesvarelser leser en fil med oversikt over studenters besvarelser (navn og format på denne filen er beskrevet i oppgave-innledningen) og returnerer en ordbok der en students brukernavn er nøkkel og filnavn med studentens besvarelse er verdi.

Oppgave 4f) 5 poeng

Utvid klassen **Oblig** med en ny metode

fordelRetting tar som parametere en ordbok med studenters besvarelser og en liste med rettere, og lar retterne vurdere én og én besvarelse (bruk metoden **vurder** i klassen **Retter**) slik at alle retterne ideelt sett vurderer like mange. Altså skal ingen retter vurdere mer enn 1 besvarelse mer enn de andre. Resultatene av retternes vurderinger samles i en ordbok der nøkkelverdiene er studentenes brukernavn. Metoden markerer obligen som rettet og returnerer ordboken med resultater.

Oppgave 4g) 15 poeng

Skriv følgende non-public metoder i klassen **Emne**:

_opprettOblig genererer et unikt oblignavn, og ber bruker på terminalen om å oppgi frist på formen ååmmdd. Deretter legges en ny oblig til emnet.

_startRetting ber om dagens dato og sjekker om noen av obligene i emnet har en utløpt frist uten at besvarelsene er rettet. I så fall henter den alle besvarelser for obligen, fordeler dem på rettere og registrerer resultatet av rettingen i Student-objekter (på riktig oblig) for studenter som har levert.

_skrivEksamensListe bygger opp en liste med brukernavn for alle studenter som har fått godkjent alle obliger registrert for emnet, og skriver til sist ut denne på terminal med en passende overskrift.

Question 18
Attached

Oppgave 4 Administrasjon av obligatoriske oppgaver (50 poeng)

Universitetet i Ruritania skal digitalisere sin håndtering av studentenes obligatoriske oppgaver («obliger»), og du skal lage deler av en pilot med et objektorientert design basert på innkapsling, dvs. instansvariable og non-public metoder (navn starter med _) skal kun benyttes i den klassen de er definert. **Det er i mange av deloppgavene nødvendig å bruke metoder fra andre klasser/ deloppgaver. Du kan bruke disse selv om du ikke har skrevet dem.**

Emnene har ulike antall obligatoriske oppgaver som kreves godkjent, og hver oblig i et emne må registreres manuelt hvert semester. En student må ha alle oppgaver gitt i emnet godkjent for å få gå opp til eksamen, men i denne piloten foretas det ikke kontroll av om studenten har fått godkjent forrige oblig før godkjenning av besvarelse for neste oblig. Foruten obliger har et emne registrerte studenter og ansatte rettere. Både rettere og studenter har unike brukernavn.

Innleveringen av besvarelser for en oblig foregår i et annet system som du ikke skal lage. Dette systemet lager en fil for hver oblig, med et entydig navn etterfulgt av «.txt», for eksempel *oblig1.txt*. Filen inneholder en linje per student på følgende format (eksempel med to linjer):
student1 obl1student1.txt
student2

I eksempelet er *student1* et entydig brukernavn og *obl1student1.txt* er navnet på filen der student1s besvarelse ligger. *student2* i eksempelet har ikke levert noen besvarelse på oblig 1.

Hver besvarelse skal vurderes av en retter for emnet. Når fristen for en obligatorisk oppgave er utløpt, fordeles besvarelser på retterne, som vurderer en og en til godkjent (1) eller underkjent (0).

Piloten har funksjonalitet for hente ut en eksamensliste med brukernavn på alle studenter som er kvalifisert for eksamen – dvs har godkjent alle registrerte obliger i emnet.

Oppgave 4a) 5 poeng

Definer en klasse **Emne** (inkludert konstruktør) som holder orden på og administrerer obliger, rettere og studenter i et emne. Konstruktørens parametere gir verdier til emnekode (streng), registrerte studenter (ordbok med Student-objekter) og rettere (liste med Retter-objekter).

Oppgave 4b) 5 poeng

Skriv metoden **administrer** i klassen Emne. Metoden skriver ut emnekoden for emnet og en meny på terminalen før den ber om kommando. Den tar imot følgende lovlige kommandoer fra en bruker:
O: Ny oblig
F: Frist ute, start retting
L: Lag eksamensliste
A: Avslutt

Annen input skal gi feilmelding og nytt forsøk. En kommando bør gjenkjennes selv om det er blanke foran eller bak, og uansett om brukeren skriver liten eller stor bokstav (metodene **strip** og **upper** er nyttige her). For å utføre kommandoene skal metoden kalle på følgende non-public metoder som også ligger i klassen **Emne**:
O -> **_opprettOblig**
F -> **_startRetting**
L -> **_skrivEksamensListe**
Disse metodene skal skrives i senere deloppgaver. De opererer kun på instansvariabler i klassen, og har ingen parametere (annet enn **self**) eller returverdier.

Oppgave 4c) 10 poeng

Skriv en klasse **Student** med instansvariabler for brukernavn og fullt navn (begge får verdi fra parametere til konstruktøren) og en ordbok som skal ta vare på resultater av rettinger, der obligId er nøkkel og resultatet verdien. Klassen har følgende metoder i tillegg til konstruktør:

registrer med parametere **oblig** (en streng som identifiserer en oblig entydig) og **resultat** (et heltall som angir om en besvarelse er vurdert som godkjent). Metoden registrerer hvilket resultat en student har fått på en oblig.

altGodkjent med parameter **antObliger** som angir antall obliger registrert i emnet. Metoden returnerer en boolsk verdi: **True** hvis studenten har fått alle obliger godkjent, **False** om en eller flere obliger mangler eller har et annet resultat enn godkjent (kodet som heltallet 1).

Oppgave 4d) 2 poeng

Skriv klassen **Retter** med instansvariabel retterens brukernavn (parameter til konstruktøren) og metoden:

vurder med en parameter som angir et filnavn for en besvarelse. I denne foreløpige utgaven av piloten ignorerer programmet ditt filen med besvarelsen, og returnerer alltid 1 (godkjent).

Oppgave 4e) 8 poeng

Skriv en klasse **Oblig** med tre instansvariabler: Entydig id for obligen, en leveringsfrist på formen ååmmdd (f. eks. 190906 for 6. september 2019) og om obligen er rettet eller ikke. De to første initialiseres med parametere til konstruktøren. Skriv følgende metoder i klassen Oblig:

klarForRetting med parameter som angir dagens dato. Metoden returnerer **True** hvis fristen er før dagens dato og besvarelsene for obligen ikke allerede er rettet – ellers **False**.

hentBesvarelser leser en fil med oversikt over studenters besvarelser (navn og format på denne filen er beskrevet i oppgave-innledningen) og returnerer en ordbok der en students brukernavn er nøkkel og filnavn med studentens besvarelse er verdi.

Oppgave 4f) 5 poeng

Utvid klassen **Oblig** med en ny metode

fordelRetting tar som parametere en ordbok med studenters besvarelser og en liste med rettere, og lar retterne vurdere én og én besvarelse (bruk metoden **vurder** i klassen **Retter**) slik at alle retterne ideelt sett vurderer like mange. Altså skal ingen retter vurdere mer enn 1 besvarelse mer enn de andre. Resultatene av retternes vurderinger samles i en ordbok der nøkkelverdiene er studentenes brukernavn. Metoden markerer obligen som rettet og returnerer ordboken med resultater.

Oppgave 4g) 15 poeng

Skriv følgende non-public metoder i klassen **Emne**:

_opprettOblig genererer et unikt oblignavn, og ber bruker på terminalen om å oppgi frist på formen ååmmdd. Deretter legges en ny oblig til emnet.

_startRetting ber om dagens dato og sjekker om noen av obligene i emnet har en utløpt frist uten at besvarelsene er rettet. I så fall henter den alle besvarelser for obligen, fordeler dem på rettere og registrerer resultatet av rettingen i Student-objekter (på riktig oblig) for studenter som har levert.

_skrivEksamensListe bygger opp en liste med brukernavn for alle studenter som har fått godkjent alle obliger registrert for emnet, og skriver til sist ut denne på terminal med en passende overskrift.

Question 19
Attached

Oppgave 4 Administrasjon av obligatoriske oppgaver (50 poeng)

Universitetet i Ruritania skal digitalisere sin håndtering av studentenes obligatoriske oppgaver («obliger»), og du skal lage deler av en pilot med et objektorientert design basert på innkapsling, dvs. instansvariable og non-public metoder (navn starter med _) skal kun benyttes i den klassen de er definert. **Det er i mange av deloppgavene nødvendig å bruke metoder fra andre klasser/ deloppgaver. Du kan bruke disse selv om du ikke har skrevet dem.**

Emnene har ulike antall obligatoriske oppgaver som kreves godkjent, og hver oblig i et emne må registreres manuelt hvert semester. En student må ha alle oppgaver gitt i emnet godkjent for å få gå opp til eksamen, men i denne piloten foretas det ikke kontroll av om studenten har fått godkjent forrige oblig før godkjenning av besvarelse for neste oblig. Foruten obliger har et emne registrerte studenter og ansatte rettere. Både rettere og studenter har unike brukernavn.

Innleveringen av besvarelser for en oblig foregår i et annet system som du ikke skal lage. Dette systemet lager en fil for hver oblig, med et entydig navn etterfulgt av «.txt», for eksempel *oblig1.txt*. Filen inneholder en linje per student på følgende format (eksempel med to linjer):
student1 obl1student1.txt
student2

I eksempelet er *student1* et entydig brukernavn og *obl1student1.txt* er navnet på filen der student1s besvarelse ligger. *student2* i eksempelet har ikke levert noen besvarelse på oblig 1.

Hver besvarelse skal vurderes av en retter for emnet. Når fristen for en obligatorisk oppgave er utløpt, fordeles besvarelser på retterne, som vurderer en og en til godkjent (1) eller underkjent (0).

Piloten har funksjonalitet for hente ut en eksamensliste med brukernavn på alle studenter som er kvalifisert for eksamen – dvs har godkjent alle registrerte obliger i emnet.

Oppgave 4a) 5 poeng

Definer en klasse **Emne** (inkludert konstruktør) som holder orden på og administrerer obliger, rettere og studenter i et emne. Konstruktørens parametere gir verdier til emnekode (streng), registrerte studenter (ordbok med Student-objekter) og rettere (liste med Retter-objekter).

Oppgave 4b) 5 poeng

Skriv metoden **administrer** i klassen Emne. Metoden skriver ut emnekoden for emnet og en meny på terminalen før den ber om kommando. Den tar imot følgende lovlige kommandoer fra en bruker:
O: Ny oblig
F: Frist ute, start retting
L: Lag eksamensliste
A: Avslutt

Annen input skal gi feilmelding og nytt forsøk. En kommando bør gjenkjennes selv om det er blanke foran eller bak, og uansett om brukeren skriver liten eller stor bokstav (metodene **strip** og **upper** er nyttige her). For å utføre kommandoene skal metoden kalle på følgende non-public metoder som også ligger i klassen **Emne**:
O -> **_opprettOblig**
F -> **_startRetting**
L -> **_skrivEksamensListe**
Disse metodene skal skrives i senere deloppgaver. De opererer kun på instansvariabler i klassen, og har ingen parametere (annet enn **self**) eller returverdier.

Oppgave 4c) 10 poeng

Skriv en klasse **Student** med instansvariabler for brukernavn og fullt navn (begge får verdi fra parametere til konstruktøren) og en ordbok som skal ta vare på resultater av rettinger, der obligId er nøkkel og resultatet verdien. Klassen har følgende metoder i tillegg til konstruktør:

registrer med parametere **oblig** (en streng som identifiserer en oblig entydig) og **resultat** (et heltall som angir om en besvarelse er vurdert som godkjent). Metoden registrerer hvilket resultat en student har fått på en oblig.

altGodkjent med parameter **antObliger** som angir antall obliger registrert i emnet. Metoden returnerer en boolsk verdi: **True** hvis studenten har fått alle obliger godkjent, **False** om en eller flere obliger mangler eller har et annet resultat enn godkjent (kodet som heltallet 1).

Oppgave 4d) 2 poeng

Skriv klassen **Retter** med instansvariabel retterens brukernavn (parameter til konstruktøren) og metoden:

vurder med en parameter som angir et filnavn for en besvarelse. I denne foreløpige utgaven av piloten ignorerer programmet ditt filen med besvarelsen, og returnerer alltid 1 (godkjent).

Oppgave 4e) 8 poeng

Skriv en klasse **Oblig** med tre instansvariabler: Entydig id for obligen, en leveringsfrist på formen ååmmdd (f. eks. 190906 for 6. september 2019) og om obligen er rettet eller ikke. De to første initialiseres med parametere til konstruktøren. Skriv følgende metoder i klassen Oblig:

klarForRetting med parameter som angir dagens dato. Metoden returnerer **True** hvis fristen er før dagens dato og besvarelsene for obligen ikke allerede er rettet – ellers **False**.

hentBesvarelser leser en fil med oversikt over studenters besvarelser (navn og format på denne filen er beskrevet i oppgave-innledningen) og returnerer en ordbok der en students brukernavn er nøkkel og filnavn med studentens besvarelse er verdi.

Oppgave 4f) 5 poeng

Utvid klassen **Oblig** med en ny metode

fordelRetting tar som parametere en ordbok med studenters besvarelser og en liste med rettere, og lar retterne vurdere én og én besvarelse (bruk metoden **vurder** i klassen **Retter**) slik at alle retterne ideelt sett vurderer like mange. Altså skal ingen retter vurdere mer enn 1 besvarelse mer enn de andre. Resultatene av retternes vurderinger samles i en ordbok der nøkkelverdiene er studentenes brukernavn. Metoden markerer obligen som rettet og returnerer ordboken med resultater.

Oppgave 4g) 15 poeng

Skriv følgende non-public metoder i klassen **Emne**:

_opprettOblig genererer et unikt oblignavn, og ber bruker på terminalen om å oppgi frist på formen ååmmdd. Deretter legges en ny oblig til emnet.

_startRetting ber om dagens dato og sjekker om noen av obligene i emnet har en utløpt frist uten at besvarelsene er rettet. I så fall henter den alle besvarelser for obligen, fordeler dem på rettere og registrerer resultatet av rettingen i Student-objekter (på riktig oblig) for studenter som har levert.

_skrivEksamensListe bygger opp en liste med brukernavn for alle studenter som har fått godkjent alle obliger registrert for emnet, og skriver til sist ut denne på terminal med en passende overskrift.