Informasjon
Oblig 8 (Prøveeksamen)
IN1000 våren 2026

Frist onsdag 22. april 2026, kl. 23:59

Her får du jobbe med utvalgte eksamensoppgaver fra tidligere prøveeksamener uten (så stort) tidspress. Du vil få tilgang til det samme eksamenssettet i eksamenssystemet Inspera før eksamen, så du kan bli kjent med systemet og dets begrensninger (for eksempel kan du ikke kjøre og teste koden du skriver på eksamen, men det kan du mens du jobber med obligen).

Skjemaet for eventuell egenmelding (3 dagers utsatt frist) finner du **her**.

Oppgavene

De første oppgavene er kortsvarsoppgaver som rettes automatisk på eksamen. Pass på å skrive inn nøyaktig svar med riktig type (ikke desimalpunktum eller -komma for heltall). Tekster kan skrives inn direkte (som her) eller med doble anførselstegn ("som her"), men ikke med enkle anførselstegn ('som her').

Du kan ignorere alle beskjeder av typen **Skriv ditt svar her** på oppgavene der du skal skrive programkode. Filene du skal levere i oblig 8 er følgende:

- **oblig8.txt** (svar på **alle deloppgavene til oppgave 1 + oppgave 2C og 2F**)

- **oppgave_2a.py**
- **oppgave_2b.py**

- **oppgave_2d.py**
- **oppgave_2e.py**

- **oppgave_3a.py**
- **oppgave_3b.py**
- **oppgave_3c.py**
- **oppgave_3c.py**

- **oppgave_4a.py**

- **oppgave_4b.py**
- **oppgave_4c.py**
- **oppgave_4d.py**
- **oppgave_4e.py**
- **oppgave_4f.py**
- **oppgave_4g.py**

- **oppgave_5a.py** (frivillig)
- **oppgave_5b.py** (frivillig)

Oppgave 4 er en oppgave som skal være et fullstendig (kjørbart) program når alle delene legges sammen. I denne oppgaven er det viktig at besvarelsen/koden viser hvordan **klasser** og **konstruktører** lages og hvordan **objekter opprettes,** hvordan **objekter settes inn (og tas ut) av samlinger,** og hvordan **objekter refererer til hverandre.** Besvarelser som viser at kandidaten ikke behersker disse sentrale læringsmålene vil ikke kunne få bestått på oblig 8. **På eksamen vil oppgave 4 være ca. 50% av eksamenssettet.**

Oppgave 5 er **frivillig**, men anbefales for de som sikter mot toppkarakter på eksamen. Oppgaven er vanskeligere enn resten av eksamenssettet, så dette er en gyllen mulighet til å få tilbakemelding på en utfordrende oppgave for de som sikter høyt.

Dersom du mener en programmeringsoppgave er uklar, kan du gjøre egne forutsetninger og beskrive disse. Du kan også legge til egne funksjoner eller metoder ved behov, med begrunnelse og kommentarer i koden. Videre kan du bruke løsninger fra tidligere deloppgaver selv om du selv ikke har skrevet disse. Du trenger ikke skrive kommentarer i kode som er direkte svar på en oppgave, men du kan kommentere dersom du føler det trengs en nærmere forklaring av noe du gjør.

Tillatte hjelpemidler (på eksamen)

Alle trykte og skrevne hjelpemidler. Ingen elektroniske.

Det anbefales sterkt at du ikke bruker elektroniske hjelpemidler mens du jobber med obligen, men heller lager de trykte og skrevne hjelpemidlene du vil ha med på selve eksamensdagen underveis.

1

Oppgave 1A: Tilordning

Hva er verdien til variabelen **tall** etter at følgende kode er utført?

```
tall = 4 + 3 * 2
tall = tall + 3
```

2

Oppgave 1B: Uttrykk

Hva er verdien til variabelen **tall** etter at følgende kode er utført?

```
tall = (4 + 3) * 2
tall = tall + 3
```

3 Oppgave 1C: Beslutninger

Hvilken verdi blir skrevet ut her?

```
null = 0
en = 1
to = 2
a = 0
if (null+1 > to) and (null < en):
    a = to * null
elif (null+1 > to) or (en < en):
    a = 3
else:
    a = en + 4
tall = a + 2
print(tall)
```

4 **Oppgave 1D: Løkke**

Hvilken verdi blir skrevet ut her?

```
i = 0
tall = 1
while i < 4:
    tall = tall * 2
    i = i + 1
print(tall)
```

5 **Oppgave 1E: Enda en løkke**

Hvilken verdi blir skrevet ut her?

```
i = 0
tall = 1
while i < 99:
    tall = tall * 2
    i = i + 1
print(i)
```

6 **Oppgave 1F: This is your løkky day**

Hvilken verdi blir skrevet ut her?

```
tall = [0,1,3,5,7,9]
sum2 = 0
for i in range(len(tall)):
    if tall[i] < 6:
        sum2 = sum2 + tall[i]
    else:
        sum2 = sum2 + tall[1]
print(sum2)
```

7 **Oppgave 1G: Parameteroverføring**

Hvilken verdi blir skrevet ut her?

```
def null_ut1(liste):
    ny_liste = []
    for i in liste:
        ny_liste.append(0)
    liste = ny_liste

terningkast = [1,2,3,3,5,6]

null_ut1(terningkast)
print(terningkast[2])
```

8 **Oppgave 1H: Mer parameteroverføring**

Hvilken verdi blir skrevet ut her?

```
def null_ut2(liste):
    ny_liste = []
    for i in range(len(liste)):
        liste[i] = 0

terningkast = [1,2,3,3,5,6]

null_ut2(terningkast)
print(terningkast[3])
```

9

Oppgave 1l: Dobling

Hvilken verdi blir skrevet ut her?

```
def doble (a):
    aa = a + a
    return aa

svarA = doble(3)
svarB = doble("abc")
print(str(svarA)+str(svarB))
```

10 **Oppgave 1J: Referanser**

Hvilken verdi blir skrevet ut her?

```
tall = [0,1,3,5,7,9]
liste = tall
liste.pop(2)
print(tall[2])
```

11 **Oppgave 1K: Iterering**

Hvilken verdi blir skrevet ut her?

```
heltall = 1
for tegn in "210202021122212":
    heltall = heltall * int(tegn)
print(heltall)
```

12

Oppgave 2A: Prosedyre

(OBS: Denne oppgaven baserer seg på en flere år gammel måte å få godkjent obliger til eksamen, og det er ikke slik obliger fungerer i IN1000 dette semesteret.)

Skriv en *prosedyre* **godkjent** som sjekker om en student kan framstille seg til eksamen i IN1000. Maks. poeng for obligene i IN1000 representerer vi i denne lista: [3,5,5,5,5,6,1,1]. Maks poeng er 3 for oblig 1, 5 for oblig 2, osv. På de første 6 obligene kan poengene variere fra 0 til maks, men for å få godkjent de første seks må summen være minst 19 (maks sum er 29). På de to siste gis 1 for godkjent og 0 for ikke godkjent. For at det obligatoriske arbeidet skal være godkjent må summen av poengene på oblig 1–6 være minst 19, og oblig 7 og 8 må begge ha 1 poeng. **godkjent** tar poenglista som eneste parameter og skal *skrive ut* "Obligkravet er oppfylt" om obligkravet er godkjent. Hvis det ikke er godkjent, skal prosedyren skrive ut alle kravene som ikke er ok. De tre kravene er altså:

- minst 19 poeng på oblig 1–6
- mer enn 0 poeng på oblig 7
- mer enn 0 poeng på oblig 8

Skriv prosedyren godkjent her:

13

Oppgave 2B: Funksjon

(OBS: Denne oppgaven baserer seg på en flere år gammel måte å få godkjent obliger til eksamen, og det er ikke slik obliger fungerer i IN1000 dette semesteret.)

Skriv en ***funksjon*** **godkjent** som sjekker om en student kan framstille seg til eksamen i IN1000. Maks. poeng for obligene i IN1000 representerer vi i denne lista: [3,5,5,5,5,6,1,1]. Maks poeng er 3 for oblig 1, 5 for oblig 2, osv. På de første 6 obligene kan poengene variere fra 0 til maks, men for å få godkjent de første seks må summen være minst 19 (maks sum er 29). På de to siste gis 1 for godkjent og 0 for ikke godkjent. For at det obligatoriske arbeidet skal være godkjent må summen av poengene på oblig 1–6 være minst 19, og oblig 7 og 8 må begge ha 1 poeng. **godkjent** tar poenglista som eneste parameter og returnerer **True** hvis poengkravet for å gå opp til eksamen er oppfylt, **False** ellers.

Skriv ditt svar her

14 Oppgave 2C: Grensesnitt

Gitt klassen:

```
class Student:
    def __init__(self, obliger):
        # parameteren obliger er ei liste med poeng (heltall)
        # variabler som starter med _ er utilgjengelig utenfor klassen
        self._obliger = obliger

    def antall_obliger(self):
        return len(self._obliger)

    def oblig(self, nr):
        return self._obliger[nr-1]
```

Hva (hvilke instansvariabler og metoder) er med i grensesnittet til klassen Student?

15

Oppgave 2D: Funksjon med hjemmelaget objekt

Skriv funksjonen obliger_godkjent som skal beregne akkurat det samme som funksjonen godkjent i oppgave 2B. Eneste forskjell er at parameteren til funksjonen nå er et studentobjekt av klassen Student. Merk at funksjonen skal skrives i det globale skopet og at du dermed ikke har tilgang til instansvariabelen self._obliger.

```
class Student:
    def __init__(self, obliger):
        # parameteren obliger er ei liste med poeng (heltall)
        # variabler som starter med _ er utilgjengelig utenfor klassen
        self._obliger = obliger

    def antall_obliger(self):
        return len(self._obliger)

    def oblig(self, nr):
        return self._obliger[nr-1]
```

Skriv funksjonen obliger_godkjent her:

16

Oppgave 2E: Metode

Skriv **metoden** obliger_godkjent som skal beregne akkurat det samme som funksjonen obliger_godkjent i forrige oppgave. Eneste forskjell er metoden nå skal være med i grensesnittet til klassen Student. Du trenger bare å skrive selve metoden, ikke resten av klassen.

```
class Student:
    def __init__(self, obliger):
        # parameteren obliger er ei liste med poeng (heltall)
        # variabler som starter med _ er utilgjengelig utenfor klassen
        self._obliger = obliger

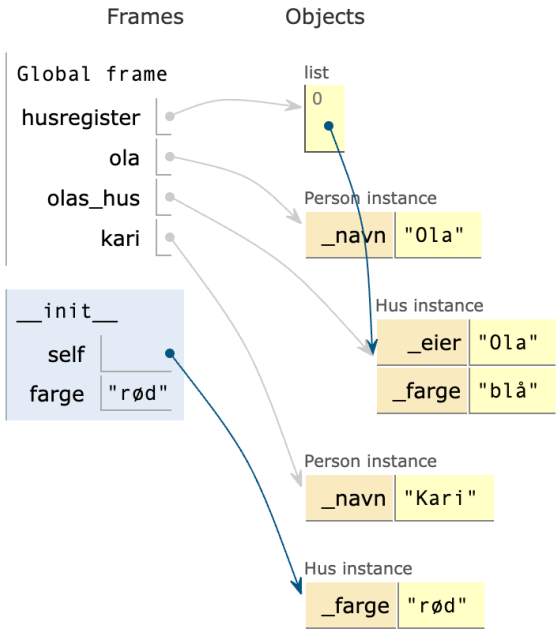
    def antall_obliger(self):
        return len(self._obliger)

    def oblig(self, nr):
        return self._obliger[nr-1]
```

Skriv metoden obliger_godkjent her:

17 Oppgave 2F: Programtilstand

Nedenfor ser du et program og en datastrukturtegning fra PythonTutor. Tegningen viser tilstanden i programmet underveis.



Hidden variables: global:Hus, global:Person

```python
class Hus:
    def __init__(self, farge):
        self._farge = farge
        # Ny instansvariabel eier som får sin verdi i metoden nedenfor
        self._eier = None

    # Nye metoder for å sette og hente navnet (str) til eieren
    def sett_eier(self, hvem):
        self._eier = hvem
    def hent_eier(self):
        return self._eier

    def mal(self, farge):
        self._farge = farge

    def hent_farge(self):
        return self._farge

class Person:
    def __init__(self, navn):
        self._navn = navn

    def hent_navn(self):
        return self._navn

# Bruker liste (list) i stedet for ordbok (dict)
# Innholdsverdien er (en referanse til) husobjektet

husregister = []
ola = Person("Ola")
olas_hus = Hus("blå")
olas_hus.sett_eier("Ola")
husregister.append(olas_hus)

kari = Person("Kari")
karis_hus = Hus("rød")
husregister.append(karis_hus)

```

Du skal ut fra tegningen finne ut hvilken linje i programmet som er den neste som skal utføres.

Skriv et linjenummer mellom 1 og 37:

18 Oppgave 3A: Fjerning av ugyldige datoer

En venn av deg har sendt deg denne klassen som vedkommende har tenkt å legge til grunn for å lage en enkel kalender:

```python
class Dato:
    def __init__(self, dd, mm, aaaa):
        self._dagnr = dd
        self._mnd = mm
        self._år = aaaa

    def dagnr(self):
        return self._dagnr
    def mnd(self):
        return self._mnd
    def år(self):
        return self._år
```

I tillegg har vennen din skrevet denne funksjonen som lager en kalender for året 2023 som ei liste av Dato-objekter:

```
def lagKalender2023():
    kalender = []
    for dag in range(32):
        for mnd in range(13):
            kalender.append(Dato(dag,mnd,2023))
    return kalender
```

Nå ber vennen din deg om å hjelpe til med å skrive en funksjon som fjerner dato-objekter for datoer som ikke finnes. Funksjonen tar inn ei liste med datoobjekter (som er det lagKalender2023 returnerer) og returnerer lista der de ugyldige datoene er fjernet. For å gjøre dette kan du legge til grunn at denne ordboka er tilgjengelig i det globale skopet:

dageriMnd = { 1:31, 2:28, 3:31, 4:30, 5:31, 6:30, 7:31, 8:31, 9:30, 10:31, 11:30, 12:31}

Skriv funskjonen fjernUgyldigeDatoer her:

19

Oppgave 3B: Generér en gyldig kalender

Skriv om funksjonen lagkalender2023, slik at den bare lager gyldige datoobjekter. **Dato**-objektene skal komme i riktig (kronologisk) rekkefølge med 1. januar 2023 først og 31. desember 2023 til slutt. Du kan fortsatt bruke denne globale ordboka:

dageriMnd = { 1:31, 2:28, 3:31, 4:30, 5:31, 6:30, 7:31, 8:31, 9:30, 10:31, 11:30, 12:31}

```
def lagKalender2023():
    kalender = []
    for dag in range(32):
        for mnd in range(13):
            kalender.append(Dato(dag,mnd,2023))
    return kalender
```

Skriv funksjonen lagkalender2023 her:

20 Oppgave 3C: Kalendere for flere år

I et år er det 365 dager, bortsett fra i skuddår som har 29. februar (29.2.) i tillegg. Så et skuddår har 366 dager. Skriv en funksjon som returnerer ei ordbok med kalendre for årene fra og med 2023 til og med 2041. En kalender er ei liste med dato-objekter (jf. kalenderen for 2023 i oppgavene foran). I tillegg til ordboka dageriMnd har du global tilgang denne lista over skuddår i perioden:

skuddår = [2024, 2028, 2032, 2036, 2040]
dageriMnd = { 1:31, 2:28, 3:31, 4:30, 5:31, 6:30, 7:31, 8:31, 9:30, 10:31, 11:30, 12:31 }

Ordboka som returneres skal ha årstallet som nøkkel, og innholdsverdien skal være kalenderen (liste av datoobjekter) for det året.

Skriv funksjonen lagKalender2023_2040 her:

i Introduksjon til oppgave 4

I filen **oblig8_oppgave4.pdf** ser du en PDF-versjon av hele oppgave 4. Det anbefales å skrive ut denne PDF-filen til oblig 8. (På selve eksamen i mai, hvor du ikke kan gjøre dette, vil du i tillegg til PDF-versjonen finne oppgaveteksten til hver deloppgave der du skal skrive besvarelsen din.)

21 Oppgave 4A: Emne

Skriv klassen **Emne** slik den er beskrevet i grensesnittet i PDF-filen (deloppgave A).

Skriv ditt svar her

22 Oppgave 4B: Studieprogram

Skriv klassen **Studieprogram** slik den er beskrevet i grensesnittet i PDF-filen (deloppgave B).

Skriv ditt svar her

23 Oppgave 4C: Hovedprogram (testprogram)

I denne deloppgaven skal du lage et hovedprogram som tester funksjonaliteten i klassene du allerede har skrevet, se deloppgave C i PDF-filen.

Skriv ditt svar her

24 Oppgave 4D: Institutt

Skriv klassen **Institutt** slik den er beskrevet i grensesnittet i PDF-filen (deloppgave D).

Skriv ditt svar her

25 Oppgave 4E: Utvidet Emne-klasse

Utvid klassen **Emne** som beskrevet i Deloppgave E i PDF-filen.

Skriv ditt svar her

26 Oppgave 4F: Utvidet Institutt-klasse

Utvid klassen **Institutt** som beskrevet i Deloppgave F i PDF-filen.

Skriv ditt svar her

27 Oppgave 4G: Utvidet hovedprogram / testprogram

Utvid hovedprogrammet ditt som beskrevet i deloppgave G i PDF-filen.

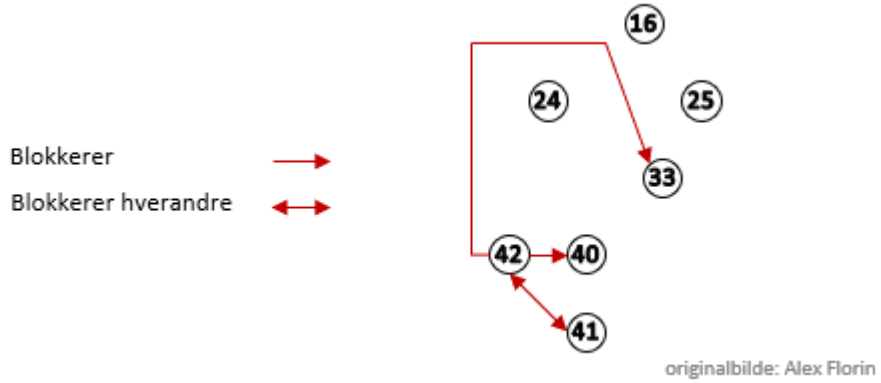
Skriv ditt svar her

28

Oppgave 5A: Blokkerte oppdrag (frivillig på oblig 8)

I brettspillet *Gloomhaven* kan man spille mange forskjellige oppdrag som har hvert sitt nummer. Når du fullfører et oppdrag, kan dette blokkere andre oppdrag (slik at disse ikke lenger kan spilles).

Et forenklet flytskjema for noen av oppdragene i spillet ser du her:



originalbilde: Alex Florin

Dette representerer vi med en ordbok **oppdrag** hvor nøklene er oppdragsnummer, og verdiene lister hvor

- Element 0 viser om oppdraget er fullført (**True**) eller ikke (**False**)
- Element 1 er en liste over oppdrag som kan blokkere dette oppdraget hvis de fullføres

Før noen av oppdragene er fullført, vil denne ordboken dermed se slik ut:

```
#           oppdragsnummer: [fullført, blokkeres_av]
oppdrag = { 16: [False, []],
            24: [False, []],
            25: [False, []],
            33: [False, [42]],
            40: [False, [42]],
            41: [False, [42]],
            42: [False, [41]]}
```

Lag en funksjon **er_blokkert(nummer, oppdrag)** som tar inn følgende parametre:

- **nummer**: heltallsnummer på et oppdrag som vi skal sjekke om er blokkert
- **oppdrag:** en ordbok på formatet vist i eksempelet over, men ikke nødvendigvis med de samme verdiene

Funksjonen skal returnere **True** hvis oppdraget med nummer **nummer** er blokkert av et annet oppdrag som er fullført, og **False** hvis ikke.

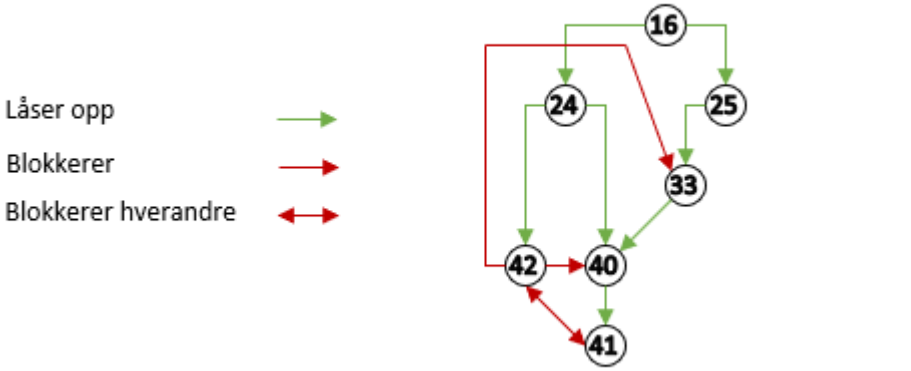
I denne deloppgaven kan du anta at **nummer** alltid vil være en nøkkel som finnes i **oppdrag**.

Skriv ditt svar her

29

Oppgave 5B: Opplåste oppdrag (frivillig på oblig 8)

I tillegg til blokkering, kreves det ofte at andre oppdrag er fullført for at et oppdrag skal låses opp og kunne spilles. Vi kan legge til dette i flytskjemaet, som da ser slik ut:



originalbilde: Alex Florin

For å representere dette utvider vi ordboken **oppdrag** fra forrige deloppgave, slik at nøklene fortsatt er oppdragsnummer, og verdiene lister hvor

- Element 0 viser om oppdraget er fullført (**True**) eller ikke (**False**)
- Element 1 er en liste over oppdrag som kan blokkere dette oppdraget hvis de fullføres
- Element 2 er listen over oppdrag som alle må fullføres for å låse opp dette oppdraget

Før noen av oppdragene er fullført, vil denne ordboken se slik ut:

```
#           oppdragsnummer: [fullført, blokkeres_av, låses_opp_av]
oppdrag = { 16: [False, [], []],
            24: [False, [], [16]],
            25: [False, [], [16]],
            33: [False, [42], [25]],
            40: [False, [42], [24, 33]],
            41: [False, [42], [40]],
            42: [False, [41], [24]]}
```

Lag en funksjon **kan_spilles(nummer, oppdrag)** som tar inn følgende parametre:

- **nummer**: heltallsnummer på et oppdrag som vi skal sjekke om kan spilles
- **oppdrag:** en ordbok på formatet vist i eksempelet over, men ikke nødvendigvis med de samme verdiene

Funksjonen skal returnere **False** hvis oppdraget ikke kan spilles, det vil si hvis minst en av følgende stemmer:

- dette oppdragsnummeret finnes ikke i ordboken
- oppdraget er blokkert, det vil si at **er_blokkert(nummer, oppdrag)** returnerer **True**
  (du kan anta at denne funksjonen virker selv om du ikke fikk til forrige deloppgave)
- man har ikke fullført alle oppdragene som dette oppdraget låses opp av

Hvis oppdraget derimot kan spilles, skal funksjonen returnere **True**.

Skriv ditt svar her