Informasjon om eksamen

UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Prøveeksamen i IN1900 høsten 2023

Tillatte hjelpemidler: Ingen
En kalkulator er tilgjengelig i Inspera.

Det er viktig at du leser denne forsiden nøye før du starter.

Eksamen består av flervalgsoppgaver og tekstoppgaver hvor du skal skrive små programmer eller tolke programmer og skrive hva som vil skrives ut på skjermen. Det er til sammen 12 oppgaver som skal besvares, og til sammen 70 tilgjengelige poeng på disse oppgavene. Arbeidsmengde og vanskelighetsgrad er omtrent den samme som til avsluttende eksamen.

Maksimalt oppnåelig poengsum er oppgitt for hver oppgave. På oppgaver med flere deloppgaver teller hver deloppgave like mye. På flervalgsoppgaver får man samme poeng (0) for feil svar som for manglende svar, så det lønner seg alltid å markere et svar.

Hvis du savner opplysninger, kan du legge dine egne forutsetninger til grunn og gjøre rimelige antagelser, så lenge de ikke bryter med oppgavens "ånd". I programmeringsoppgaver bør du da gjøre rede for forutsetningene og antagelsene du gjør, for eksempel i kommentarer til koden.

All kode i oppgavetekstene er skrevet i Python 3.

De fleste oppgavene gir kort kode med lite behov for kommentarer, med mindre man gjør noe komplisert eller ikke-standard (anbefales ikke; men i så fall skal kommentarene forklare ideene bak program-konstruksjonene slik at det blir lett å vurdere koden).

En oppgave kan be deg skrive en funksjon. Et hovedprogram der man kaller funksjonen er da ikke påkrevd, med mindre det er angitt.

1 Reversere en liste

Vi ønsker å reversere listen **L** slik at listen **rev** inneholder de samme elementene, men i motsatt rekkefølge. Vi har to alternative forslag til kode for å løse denne oppgaven.

Alternativ 1:

```
rev = [ ]
for i in range(len(L)):
  rev[i] = L[len(L) - i]
```

Alternativ 2:

```
rev = [L[len(L)-k] for k in range(len(L))]
```

Hvilke(t) alternativ er riktig?

Velg ett alternativ:

- ○ Alternativ 1 er riktig
- ○ Alternativ 2 er riktig
- ○ Begge er riktige
- ○ Ingen er riktige ✔

Maks poeng: 2

2 Indeksering i lister

Vi har følgende tre lister:

- **kort = [1, 0, -1]**
- **lang = [1, 2, 3, 4, 5]**
- **liste = [1, 2, [1], [2], [[1]], [[2]]]**

Til venstre for tabellen står det fem uttrykk, og på toppen av tabellen står det seks mulige resultater når uttrykkene evalueres. Velg riktig svaralternativ for hvert uttrykk.

	1	2	[1]	[2]	[[1]]	[[2]]
liste[-2]	○	○	○	○	✔	○
liste[kort[1]]	✔	○	○	○	○	○
liste[kort[2] * lang[2]]	○	○	○	✔	○	○
liste[lang[kort[0]]]	○	✔	○	○	○	○
liste[liste[3][0]]	○	○	✔	○	○	○

Maks poeng: 2.5

3 Hva skrives ut?

Hva skrives ut i terminalvinduet når følgende kode kjøres?

```
xlist = [-2, -1, 0, 1, 2]
ylist = []
for x in xlist:
    ylist.append(xlist[x])
print(ylist[4])
```

Velg ett alternativ

- 4
- [4]
- 0 ✔
- [0]
- 2
- [2]

Maks poeng: 2

4 Funksjoner med parametre

Vi ønsker å lage en klasse som implementerer denne funksjonen:

$$f(x; a, b, c) = ax^2 + bx + c$$

Funksjonen har tre parametre (a, b, og c). Det skal være mulig å sette verdiene til de tre parametrene først, og deretter skal det være mulig å kalle på funksjonen ved bare å skrive **f(x)**. Her er et forslag til løsning:

```
from math import sqrt

class Quadratic:
   def __init__(self, a, b, c):
       self.a, self.b, self.c = a, b, c

   def __call__(self, x):
      return self.a**self.x*2 + self.b*self.x + self.c

f = Quadratic(2, 5, 3)
print(f(x))
```

Dessverre virker ikke programmet slik det skal. Hvilke(n) feil har sneket seg inn?

Skriv ditt svar her

Maks poeng: 3

5 Feilhåndtering (exceptions)

Vi har laget følgende program:

```
import sys
x = [3, 2, 1, 0]
try:
    v1 = int(sys.argv[1])
    v2 = int(sys.argv[2])
    svar = x[v1] / x[v2]
except ValueError:
    print("Error A")
    sys.exit()
except IndexError:
    print("Error B")
    sys.exit()
except ZeroDivisionError:
    print("Error C")
    sys.exit()
```

Vi prøver å kjører programmet fra kommandolinjen med hvert av alternativene vist nedenfor (på venstre side av tabellen). Avgjør for hver rad i tabellen hva som skrives ut (hvor "Ingen utskrift" betyr at det ikke skrives ut noe).

Finn de som passer sammen:

	Error A	Error B	Error C	Ingen utskrift
python divide.py 3	○	○ ✔	○	○
python divide.py v1 v2	○ ✔	○	○	○
python divide.py 0 0	○	○	○	○ ✔
python divide.py 1 1 0 3	○	○	○	○ ✔
python divide.py 0 -3	○	○	○ ✔	○

Maks poeng: 2.5

6 Summer og testfunksjoner

Den naturlige logaritmen gitt ved $\ln(1-x)$ har følgende rekkeutvikling:

$$\ln(1-x) = -x - \tfrac{1}{2}x^2 - \tfrac{1}{3}x^3 - \tfrac{1}{4}x^4 + \cdots$$

a) Skriv en Python-funksjon **log_1minus(x,n)** som for gitt **x** og **n** beregner og returnerer summen ovenfor, og som tar med alle ledd med grad $\leq n$.

b) Skriv en testfunksjon for **log_1minus(x,n)**. Testfunksjonen skal sammenlikne svaret fra **log_1minus(x,50)** med svaret fra funksjonen **log(1-x)** i modulen **math**. Vi antar her at den sistnevne funksjonen gir det korrekte svaret. Testen skal gjøres for tre ulike **x**-verdier i det åpne intervallet (0,1) som du selv velger. Du kan anta at for **n=50** er nøyaktigheten til rekkeutviklingen minst 10 desimaler.

Skriv ditt svar her

```
1
```

Maks poeng: 6

7 Implementere et folkeregister

(Uendret fra høst 2022)

Regjeringen i landet Ruritania har et folkeregister som for hver innbygger inneholder fire opplysninger: fødselsnummer, navn, yrke og adresse. Folkeregisteret ligger på en tekstfil *register.txt* med fire kolonner og en overskrift, og hvor de tre første linjene kan se slik ut (hele filen er mye lenger):

Fnr	Navn	Yrke	Adresse
10125564233	Ole Olsen	Lege	Gåsemyrgaten 14, 0323 Gåseby
30069912345	Signe Nes	Lærer	Gufsealleen 5, 6233 Olsby

Alle oppføringer i en kolonne starter i nøyaktig samme tegnposisjon som den tilhørende overskriften. En gang i måneden skal filen oppdateres slik at personer som ikke lenger skal stå i registeret fjernes (f.eks. hvis noen har dødd siste måneden) og slik at nye personer registreres (f.eks. hvis noen har blitt født eller har flyttet inn i landet siste måneden).

Din oppgave er å lage et Python-program som gjør denne oppdateringen av filen. I praksis gjennomføres dette ved at tekstfilen først leses inn i et dictionary med hvert fødselsnummer som en nøkkel og resten av opplysningene som tilhørende verdi. Så gjøres de nødvendige endringer (fjerne personer og legge til personer fra registeret) på dette dictionariet, og til slutt skrives hele dictionariet ut til fil igjen. Personer som skal legges til registeret ligger i en egen fil *newpersons.txt* som har akkurat samme format som den ovennevnte filen *register.txt*. Personer som skal fjernes fra registeret ligger i en egen fil *remove.txt* og denne består bare av en enkelt kolonne med fødselsnummerne til de som skal fjernes fra folkeregisteret.

a) Skriv funksjonen **read_register()** som leser inn hele filen *register.txt* til et dictionary som forklart over. Funksjonen skal til slutt returnere denne dictionarien.

b) Skriv funksjonen **add_to(register)** hvor argumentet **register** er et dictionary som er laget med funksjonen i punkt a. Funksjonen skal lese filen *newpersons.txt* og gjøre de nødvendige endringer i dictionarien. Funksjonen skal til slutt returnere det oppdaterte dictionariet.

c) Skriv funksjonen **remove_from(register)** hvor argumentet **register** er et dictionary som er laget med funksjonen i punkt a. Funksjonen skal lese filen *remove.txt* og gjøre de nødvendige endringer i dictionarien. Funksjonen skal til slutt returnere det oppdaterte dictionariet.

d) Skriv funksjonen **write_register(register)** som skriver dictionariet **register** til filen *register.txt*. Du kan anta at den eksisterende filen med samme navn ikke er skrivebeskyttet, slik at programmet bare overskriver denne.

Skriv ditt svar her

```
1
```

Maks poeng: 8

8 Beregne verdien til et polynom

Fibonnacci-polynomer er definert på denne måten:

$$F_0(x) = 0$$
$$F_1(x) = 1$$
$$F_n(x) = xF_{n-1}(x) + F_{n-2}(x) \quad \text{for } n = 2, 3, \ldots$$

De første fem polynomene i følgen er
$F_0(x) = 0,\ F_1(x) = 1,\ F_2(x) = x,\ F_3(x) = x^2 + 1,\ F_4(x) = x^3 + 2x$.
Generelt vil $F_n(x)$ være et polynom av grad $n - 1$ for $n \geq 1$.

Skriv en Python-funksjon **fibonnaccipoly(n, x)** som beregner verdien til $F_n(x)$ i et oppgitt punkt **x**.

Hint: Hvis du har regnet ut $F_{n-1}(x)$ og $F_{n-2}(x)$ er det lett å regne ut $F_n(x)$ fra formelen over. Lag en løkke som enten lagrer alle verdiene $F_0(x), F_1(x), F_2(x), \ldots$ i en liste, eller bare lagrer de to siste verdiene og oppdaterer disse for hver iterasjon.

Merk: Oppgaven skal løses ved å bruke en løkke. Det er også mulig å løse oppgaven med såkalt rekursjon, men dette er ikke pensum i kurset og det er ikke dette vi er ute etter.

Skriv ditt svar her

```
1
```

Maks poeng: 5

9 Representere polynom som klasse

Du skal i denne oppgaven lage en klasse for å representere Fibonnacci-polynomet $F_n(x)$ som definert i forrige oppgave. Klassen skal ha følgende navn og struktur:

```
class FibonnacciPoly:
    def __init__(self, n):
        # Her mangler det noe

    def __str__(self):
        # Her mangler det noe

    def __call__(self, x):
        # Her mangler det noe
```

Klassen over skal kunne brukes på denne måten:

```
f = FibonnacciPoly(4)     # Instans av klassen som representerer F_4(x)
print(f)                  # UTSKRIFT:    Fibonnacci polynomial: n = 4
print(f(2))               # Evaluer F_4(x) for x=2        UTSKRIFT:    23
```

a) Skriv ferdig konstruktøren **__init__(self, n)**.

b) Skriv ferdig metoden **__str__(self)**. Denne skal returnere en streng som forteller at dette et Fibonnacci-polynom og hvilken verdi N har (som i eksempelet over).

c) Skriv ferdig metoden **__call__(self, x)**. Denne skal regne ut og returnere verdien til Fibonnacci-polynomet $F_n(x)$ for den oppgitte verdien av **x**.

d) Skriv et program som bruker klassen **FibonnacciPoly** til å skrive ut de 100 første Fibonnacci-tallene, som er gitt ved $F_n(1)$ for $n < 100$.

Skriv ditt svar her

```
1
```

Maks poeng: 12

10 Klasse for 2x2-matriser

En 2x2-matrise kan ses på som en tabell med to rader og to kolonner:

$$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Radene kan representeres som vektorer:

$$r_1 = [a \quad b], r_2 = [c \quad d]$$

Det kan kolonnene også:

$$c_1 = \begin{bmatrix} a \\ c \end{bmatrix}, c_2 = \begin{bmatrix} b \\ d \end{bmatrix}$$

En klasse som representerer en slik matrise kan implementeres som følger:

```
import numpy as np

class Matrix2x2:
    def __init__(self, a, b, c, d):
        # rader
        self._row1 = np.array([a, b])
        self._row2 = np.array([c, d])
        # kolonner
        self._column1 = np.array([a, c])
        self._column2 = np.array([b, d])

    def __str__(self):
        return str(self._row1) + "\n" + str(self._row2)
```

Vi ønsker å utvide denne klassen til å støtte matrisemultiplikasjon, som er definert slik:

$$A \cdot B = \begin{bmatrix} r_1 \cdot c_1 & r_1 \cdot c_2 \\ r_2 \cdot c_1 & r_2 \cdot c_2 \end{bmatrix}$$

det vil si at hvert element blir et prikkprodukt mellom en radvektor og en kolonnevektor.
Prikkproduktet kan regnes ut ved hjelp av funksjonen **np.dot(v1, v2)** dersom **v1** og **v2** er numpy-arrays med samme lengde (som da altså representerer to vektorer).

a) Hvilken spesialmetode må du legge til i klassen **Matrix2x2** for å støtte denne typen bruk?

```
A = Matrix2x2(1, 2, 3, 4)
I = Matrix2x2(1, 0, 0, 1)
print(A*I)
```

b) Forklar (uten å skrive kode) hvilke parametre denne metoden må ta, og hva den skal returnere.

c) Skriv metoden som implementerer denne utvidelsen av klassen.

Skriv ditt svar her

```
1
```

Maks poeng: 9

11 Differenslikninger

(Uendret fra høst 2022)

Vi har følgende system av differenslikninger for $n \geq 0$:

$$\begin{aligned} x_{n+1} &= x_n + y_{n+1} \\ y_{n+1} &= y_n + k\,x_n(x_n - 1) \end{aligned}$$

Systemet kalles *Bogdanov-avbildningen* etter den russiske matematikeren Rifkat Bogdanov.

a) Skriv en funksjon **bogdanov(N, k, x0, y0)** som beregner løsningen (x_n, y_n) på differenslikningene over for $n = 1, 2, \ldots, N$. De tre siste argumentene til funksjonen er parameteren *k* og initialbetingelsene x_0 og y_0. Funksjonen skal returnere løsningen som et tuppel (x, y) hvor x er en liste (eller array) med løsningsverdiene $x_0, x_1, x_2, \ldots, x_N$ og y er en liste (eller array) med løsningsverdiene $y_0, y_1, y_2, \ldots, y_N$.

b) Vi kan ikke avgjøre numerisk om en uendelig tallfølge $x_0, x_1, x_2, \ldots$ konvergerer eller ikke. Men vi kan få en pekepinn ved å regne ut de N+1 første verdiene i tallfølgen og sjekke om de to siste verdiene vi har regnet ut er tilnærmet like (det kan tyde på at følgen nærmer seg konvergens). Skriv en Python-funksjon **convergence(x, eps)** som bruker dette kriteriet til å få en pekepinn på om en tallfølge konvergerer. Her er x en liste med de første verdiene i en tallfølge og eps er en toleranseverdi. Funksjonen skal returnere True hvis de to siste verdiene i listen ligger nærmere hverandre enn toleransen eps og skal returnere False i motsatt fall.

c) Forklar kort hvordan funksjonen du skrev i punkt b) kan brukes til å få en pekepinn på om løsningen som returneres fra funksjonen **bogdanov** i punkt a) konvergerer.

d) Noen differenslikninger har en løsning som ikke konvergerer, men som bare sykler gjennom (repeterer i det uendelige) et endelig antall verdier. For eksempel sykler tallfølgen 0,1,0,1,0,1,0,1, ... gjennom verdiene 0,1 mens tallfølgen 0,1,2,0,1,2,0,1,2,... sykler gjennom verdiene 0,1,2. Lengden på sykelen er henholdsvis 2 og 3 i disse eksemplene. Generelt kan en sykel ha lengde k > 0. Det betyr at de første k verdiene i følgen $x_0, x_1, x_2, \ldots, x_{k-1}$ er identisk med de neste k verdiene $x_k, x_{k+1}, x_{k+2}, \ldots, x_{2k-1}$ (altså $x_0 = x_k$, $x_1 = x_{k+1}$, ..., $x_{k-1} = x_{2k-1}$) som igjen er lik de neste k verdiene $x_{2k}, x_{2k+1}, x_{2k+2}, \ldots, x_{3k-1}$, osv. Anta at vi allerede har skrevet en funksjon **isCycle(x,k)** som sjekker om følgen x (en liste) har en sykel av lengde k (et positivt heltall) og som isåfall returnerer True (og False ellers). Bruk denne til å skrive en ny funksjon **findCycle(x, kmax)** som finner den minste verdien av k i intervallet $1, 2, \ldots, kmax$ som er slik at x sykler gjennom k verdier. Hvis funksjonen finner en slik k skal den returnere denne verdien. Hvis funksjonen ikke finner noen slik k, skal funksjonen returnere 0.

Skriv ditt svar her

```
1
```

Maks poeng: 12

12 Planeters bane rundt sola

(Uendret fra høst 2022)

Vi ser i denne oppgaven på bevegelsen til en planet rundt sola. Anta at planeten på tidspunkt t har posisjon $(x(t), y(t), z(t))$ og hastighetsvektor $(u(t), v(t), w(t))$. Da kan en med Newtons lover vise at bevegelsen er styrt av disse seks differensiallikningene:

$$
\begin{aligned}
x'(t) &= u(t) \\
y'(t) &= v(t) \\
z'(t) &= w(t) \\
u'(t) &= -x(t)/(x(t)^2 + y(t)^2 + z(t)^2)^{3/2} \\
v'(t) &= -y(t)/(x(t)^2 + y(t)^2 + z(t)^2)^{3/2} \\
w'(t) &= -z(t)/(x(t)^2 + y(t)^2 + z(t)^2)^{3/2}
\end{aligned}
$$

Lag et program i Python som løser dette ODE-systemet ved hjelp av ODESolver-modulen (se vedlagt fil) når initialbetingelsene er $x(0) = y(0) = z(0) = u(0) = v(0) = w(0) = 1$. Programmet skal benytte RungeKutta4-metoden med tidspunkter gitt ved

time_points = np.linspace(0,50,5001).

hvor vi antar at numpy er importert med **import numpy as np**. Løsningen skal skrives ut på skjerm som en tabell med fire kolonner og med en rad for hvert løsningstidspunkt (se eksempel under). Tidspunkter skal skrives ut med to desimaler, mens øvrige verdier skal skrives ut med 4 desimaler. Sørg for at tabellen blir pent formatert ved at tall i samme kolonne er venstrejustert i forhold til hverandre (dvs starter i samme posisjon). Merk at tabellen kun skriver ut tid og posisjon, mens hastighetsvektoren ikke skal tas med i utskriften.

Time	x(t)	y(t)	z(t)
0.00	1.0000	1.0000	1.0000
0.01	...	...	...

Det er ikke nødvendig å skrive kode for å plotte løsningen som kurver. De relevante delene av pakken ODESolver finnes i vedlagte pdf-fil.

Skriv ditt svar her

```
1
```

Maks poeng: 6

Question 12
Attached

```
import numpy as np

class ODESolver:
    def __init__(self, f):
        self.f = lambda u, t: np.asarray(f(u, t), float)

    def set_initial_condition(self, U0):
        if isinstance(U0, (float,int)):  # scalar ODE
            self.neq = 1                 # no of equations
            U0 = float(U0)
        else:                            # system of ODEs
            U0 = np.asarray(U0)
            self.neq = U0.size           # no of equations
        self.U0 = U0

    def solve(self, time_points):
        self.t = np.asarray(time_points)
        N = len(self.t)
        if self.neq == 1:  # scalar ODEs
            self.u = np.zeros(N)
        else:              # systems of ODEs
            self.u = np.zeros((N,self.neq))

        self.u[0] = self.U0

        for n in range(N-1):
            self.n = n
            self.u[n+1] = self.advance()
        return self.u, self.t

class ForwardEuler(ODESolver):
    def advance(self):
        u, f, n, t = self.u, self.f, self.n, self.t
        dt = t[n+1] - t[n]
        unew = u[n] + dt*f(u[n], t[n])
        return unew

class RungeKutta4(ODESolver):
    def advance(self):
        u, f, n, t = self.u, self.f, self.n, self.t
        dt = t[n+1] - t[n]
        dt2 = dt/2.0
        k1 = f(u[n], t[n])
        k2 = f(u[n] + dt2*k1, t[n] + dt2)
        k3 = f(u[n] + dt2*k2, t[n] + dt2)
        k4 = f(u[n] + dt*k3, t[n] + dt)
        unew = u[n] + (dt/6.0)*(k1 + 2*k2 + 2*k3 + k4)
        return unew
```