Informasjon om eksamen

UNIVERSITETET I OSLO
Det matematisk-naturvitenskapelige fakultet
Skriftlig eksamen i IN1900
2021 Høst

Varighet: 14.12.21, 09:00 til 14.12.21, 13:00

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 15 oppgaver som skal besvares, og til sammen 75 tilgjengelige poeng på disse oppgavene. Oppgave 16 skal ikke besvares, men brukes av sensorene under sensuren for å legge inn poeng fra midtveiseksamen.

Maksimalt oppnåelig poengsum er oppgitt for hver oppgave. På oppgaver med flere deloppgaver teller hver deloppgave like mye.

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 Løkke som teller

Hvilken verdi har variabelen n etter at disse programsetningene er utført?

```
n = 0
k = 10
while n < k:
    n = n + 2
    k = k + 1
```

Velg ett alternativ:

- 10
- 12
- 14
- 16
- 18
- 20 ✔
- 22
- 24

Maks poeng: 2

2 Hvilke hører sammen?

Anta at vi har listen **x = [[-1],[1], 1, 2, 0].** Til venstre for tabellen står det syv uttrykk, og på toppen av tabellen står det seks mulige resultater når uttrykkene evalueres. Velg riktig svaralternativ for hvert uttrykk.

Finn de som passer sammen:

	0	1	[-1]	[1]	[[-1]]	[[1]]
x[-1]	✔					
x[x[-1]]			✔			
x[x[2]-1]			✔			
x[x[2]:x[3]]						✔
x[2:3]				✔		
x[1:2]						✔
x[x[0][0]]	✔					

Maks poeng: 3.5

3 Hvilken linje mangler?

Funksjonen **diff2nd(f,x,h)** finner en approksimasjon til den annenderiverte til en funksjon f i punktet x:

```
def diff2nd(f, x, h=1E-6):
    r = (f(x-h) - 2*f(x) + f(x+h))/(h*h)
    return r
```

I koden under ønsker vi å bruke funksjonen over til å estimere den annenderiverte til $f(x) = 2x^3 - 3x$ for x = 1.5, og til å studere hvordan estimatet forandrer seg når vi endrer verdien av h.

```
for k in range(1,8):
    h = 10**(-k)
    #Missing line goes here
    print(f'h={h}: {d2g}')
```

En linje mangler i koden. Hvilken av følgende linjer må legges til for at koden skal gjøre det vi ønsker?

Velg ett alternativ

- d2g = diff2nd(lambda x: 2*x**3-3*x, 1.5, h) ✔
- d2g = diff2nd(2*x**3-3*x, 1.5, h)
- f = diff2nd(f=2*x**3-3*x, 1.5, h)
- d2g = diff2nd(f=2*x**3-3*x, 1.5)

Maks poeng: 2

4 Funksjon med parametre

Hva skrives ut på skjermen når dette programmet blir kjørt?

```
class F:
    def __init__(self, alpha, beta):
        self.alpha = alpha
        self.beta = beta

    def __call__(self, x):
        return self.alpha + self.beta * x

    def __add__(self, obj):
        return F(self.alpha+obj.alpha, self.beta+obj.beta)

a = F(0,1)
b = F(2,3)
c  = (a + b)(2)
print(c)
```

Velg ett alternativ:

- En feilmelding
- c
- 2
- 8
- 10 ✔
- 12
- 18

Maks poeng: 2

5 Logiske uttrykk

Avgjør for hvert av de logiske uttrykkene på venstre side av tabellen hva svaret blir når de evalueres.

Hva er verdien til uttrykkene?

	False	True
True or False and True	○	○ ✔
True and (not False or True)	○	○ ✔
not (False and True)	○	○ ✔
[e for e in [False,True] if not e][-1]	○ ✔	○
not ((2==3) and not(2==4))	○	○ ✔

Maks poeng: 2.5

6 Finne største verdi

Vi har en liste x med tall (f.eks. x = [-5, -2, -6, -3, -8, -3, -5, -16]) og ønsker å finne den største verdien i denne listen. Hvilke av kodealternativene til venstre resulterer i at y blir lik den største verdien i x? (svar "Ja" for de kodealternativene du mener resulterer i at y blir lik den største verdien i x, og svar "Nei" for de kodealternativene du mener ikke resulterer i at y blir lik den største verdien i x).

Finn de som passer sammen:

	Nei	Ja
Alternativ C	○	○ ✔
Alternativ F	○	○ ✔
Alternativ G	○	○ ✔
Alternativ D	○ ✔	○
Alternativ B	○	○ ✔
Alternativ E	○	○ ✔
Alternativ A	○ ✔	○

Maks poeng: 3.5

7 Feilhåndtering (exceptions)

Følgende program mangler en programlinje:

```
import sys
try:
    # Her mangler det en programlinje
except IndexError:
    print("Feil A")
    sys.exit()
except ValueError:
    print("Feil B")
    sys.exit()
except ZeroDivisionError:
    print("Feil C")
    sys.exit()
```

Vi erstatter den manglende programlinjen med en av alternativene på venstre side av tabellen under, og kjører programmet fra terminalvinduet ved å gi kommandoen

Terminal> python check.py 4 0

Avgjør for hver rad i tabellen hva som skrives ut fra programmet.

Finn de som passer sammen:

	Feil A	Feil B	Feil C	Ingen utskrift
j = sys.argv[1]	○	○	○	○ ✔
j,k = sys.argv[2], sys.argv[3]	○ ✔	○	○	○
j = float(sys.argv[0])	○	○ ✔	○	○
h = 1/int(sys.argv[2])	○	○	○ ✔	○
k = sys.argv[:3]	○	○	○	○ ✔

Maks poeng: 2.5

8 Numerisk derivasjon

Den deriverte til en funksjon $f(x)$ kan estimeres med en midtpunktsdifferanse:

$$f'(x) \approx \frac{f(x+\Delta x)-f(x-\Delta x)}{2\Delta x}$$

hvor $\Delta x > 0$ er en konstant.

Skriv en Python-funksjon **midpoint(f, x0, dx=1e-4)**, som bruker denne formelen til å estimere den deriverte til funksjonen f(x) i punktet x0. Funksjonen skal returnere den estimerte verdien. Inkluder også kode for å kalle på funksjonen for å estimere den deriverte til *sin(x)* i punktet $x = \pi/4$. Ta med nødvendig import.

Skriv ditt svar her

```
1
```

Maks poeng: 3

9 Beregning av en sum

Den inverse sinusfunksjonen $\arcsin x$ kan approksimeres slik:

$$\arcsin x \approx \sum_{n=0}^{N} \frac{(2n)!}{2^{2n}(n!)^2} \frac{x^{2n+1}}{2n+1}$$

hvor $n!$ betyr fakultet av n.

a) Skriv en Python-funksjon **arcsin_approx(x,N)** som for gitt x og N finner summen ovenfor og returnerer denne. Bruk funksjonen **factorial(n)** i modulen **math** til å beregne fakultet (n!).

b) For å se hvor god approksimasjon funksjonen over gir, ønsker vi å sammenlikne svaret vi får med svaret vi får når vi bruker den innebygde funksjonen **asin(x)** i modulen **math**. Skriv et program som bruker funksjonen fra deloppgave a) til å approksimere den inverse sinusfunksjonen for x = 0.5 og N=1, 2, ..., 20 og som skriver ut en tabell med tre kolonner: **N**, **arcsin_approx(x,N)** og feilen **abs(asin(x) - arcsin_approx(x,N)**).

Skriv ditt svar her

```
1
```

Maks poeng: 6

10 Klasse for tilfeldige tall

Vi kan ikke få datamaskinen til å generere virkelig tilfeldige tall, men det finnes mange såkalte *pseudo random number generators* (PRNGs) som kan generere sekvenser med tall som i praksis kan antas å være tilfeldig generert. Funksjonen under implementerer en slik PRNG. Argumentet n angir hvor mange tilfeldige tall vi ønsker å få generert, mens de resterende argumentene seed, m, a og b kan brukes til å endre måten algoritmen oppfører seg på.

```
def rand_gen(n, seed, m, a, b):
    res = [ ]
    x = seed
    for i in range(n):
        x = (a * x + b) % m
        res.append(x/(m-1))
    return res
```

a) Vi ønsker å implementere tilfeldig-tall generatoren ovenfor som en funksjon av n alene, og med seed, m, a og b som parametre til funksjonen. Gjør dette ved å implementere tilfeldig-tall generatoren som en klasse **RandomGenerator** med to spesialmetoder, slik at følgende kode fungerer:

```
rand = RandomGenerator(seed=0.01, m=2**16+1, a=75, b=74)
x = rand(50)   # Skal generere en liste med 50 tilfeldige tall
```

b) Utvid klassen **RandomGenerator** med en ny spesialmetode slik at følgende kode fungerer:

```
rand = RandomGenerator(seed=0.01, m=2**16+1, a=75, b=74)
print(rand)   # Skriver ut verdien til de fire parametrene seed, m, a og b
```

Skriv ditt svar her

```
1
```

Maks poeng: 6

11 DNA-sekvenser

En DNA-sekvens er en tekststreng av vilkårlig lengde som kun inneholder de fire tegnene A, C, G, T. For eksempel er "CGTACGCCGA" en DNA-sekvens. Du skal i denne oppgaven lage et program som analyserer slike DNA-sekvenser. DNA-sekvensen ligger i utgangspunktet på en tekstfil og må leses inn fra denne. En slik fil inneholder kun en enkelt DNA-sekvens, men for lesbarhetens skyld er sekvensen vanligvis delt opp i flere nummererte linjer. Eksempel på innhold i en slik fil (NB: tallene til venstre er også med i filen):

```
1       CGTACGCCGACGTACGCCGA
2       CGCCGACGTACGCGCCGACG
3       GTACGCGCCGACGGTACGCG
```

I dette konkrete tilfellet var DNA-sekvensen delt opp i tre linjer, men antall linjer vil variere fra fil til fil. Hver linje består av et linjenummer, etterfulgt av ett eller flere blanke tegn, og til slutt en del av DNA-sekvensen. Hele DNA-sekvensen fås ved å skjøte sammen (konkatenere) alle delsekvensene i den rekkefølgen de ligger i filen.

a) Skriv en funksjon **read_DNA(filename)** som leser en DNA-sekvens fra en fil med formatet over, og som returnerer en tekststreng som inneholder DNA-sekvensen. Du kan anta at filen har det korrekte formatet. Husk at linjenummerne og de blanke tegnene på hver linje ikke skal være med i DNA-sekvensen som returneres.

b) Skriv en funksjon **find_CG(dna)** som tar en tekststreng med en DNA-sekvens som input, og som returnerer en liste med alle posisjonene i tekststrengen hvor sekvensen "CG" forekommer. Hvis for eksempel **dna** er sekvensen "CGATCGCG" så skal funksjonen returnere listen [0, 4, 6], og hvis **dna** er sekvensen "AAT" så skal funksjonen returnere den tomme listen [].

c) Skriv en testfunksjon **test_find_CG()** for funksjonen i forrige punkt. Testfunksjonen skal teste at **find_CG** returnerer riktig svar for de to sekvensene "TATACG" og "CGCGCG".

d) Når sekvensen "CG" forekommer k ganger rett etter hverandre i en DNA-sekvens, kaller vi det for en CG-repetisjon av lengde k. For eksempel inneholder sekvensen "CGAACGCGCG" en CG-repetisjon av lengde 1 og en CG-repetisjon av lengde 3. Skriv en funksjon **longest_CG_repetition(dna)** som tar en tekststreng med en DNA-sekvens som input, og som returnerer lengden av den lengste CG-repetisjonen i sekvensen. For eksempel skal funksjonen returnere verdien 3 for DNA-sekvensen "CGAACGCGCG" og verdien 0 for DNA-sekvensen "AAGCTCA".

Skriv ditt svar her

```
1
```

Maks poeng: 12

12 Nullpunkter

Anta at vi numerisk ønsker å finne et nullpunkt til en gitt funksjon f, dvs vi ønsker å finne x slik at $f(x) = 0$. Sekant-metoden gjør dette ved å løse følgende differenslikning for $n = 2, 3, \ldots, N$:

$$x_n = \frac{f(x_{n-1})x_{n-2} - f(x_{n-2})x_{n-1}}{f(x_{n-1}) - f(x_{n-2})}$$

der x_0 og x_1 er initialbetingelser. Hvis vi tror at løsningen ligger på intervallet $[0, 5]$ så kan vi for eksempel velge $x_0 = 0$ og $x_1 = 5$.

a) Skriv en funksjon **secant(f, x0, x1, N)** som implementerer sekant-metoden for en gitt funksjon f, med initialbetingelser x0 og x1, og hvor N angir øvre grense som angitt ovenfor. Funksjonen skal returnere den siste beregnede verdien x_N.

b) I praksis vet vi ikke hvor mange ledd som må regnes ut ovenfor (dvs hvor stor N bør være) for å oppnå $f(x_N) \approx 0$. Det er ofte mer nyttig å angi en toleranseverdi $\varepsilon > 0$ og deretter regne ut akkurat så mange ledd N i differenslikningen over at vi oppnår $|f(x_N)| < \varepsilon$. Skriv en funksjon **secant_tol(f, x0, x1, eps, Nmax)** som gjør dette. Her er f funksjonen vi skal finne nullpunkt til, x0 og x1 er initialbetingelsene, eps er toleranseverdien og Nmax er maksimalt antall ledd vi skal regne ut i følgen x_n. Funksjonen skal returnere den sist beregnede verdien i følgen uansett om denne oppfyller toleransekravet eller ikke. Merk: funksjonen du skal lage skal altså regne ut *nøyaktig så mange ledd* i differenslikningen at toleransekravet oppfylles, men dersom toleransekravet *fortsatt* ikke er oppfylt etter at Nmax ledd er regnet ut, skal funksjonen returnere verdien x_{Nmax}.

Skriv ditt svar her

```
1
```

Maks poeng: 6

13 ODE-system for været

Vi skal i denne oppgaven se på et system av differensiallikninger som kalles Lorenz-modellen. Modellen er ment å gi en ekstremt forenklet beskrivelse av været og spesielt modellere den uforutsigbare oppførselen til været. Modellen består av tre funksjoner $x(t)$, $y(t)$ og $z(t)$ hvor $t \geq 0$. Litt forenklet er $x(t)$ et mål på hvor stor bevegelse det er i atmosfæren, mens $y(t)$ og $z(t)$ beskriver den horisontale og vertikale temperaturfordelingen i atmosfæren. Følgende system av differensiallikninger beskriver hvordan $x(t)$, $y(t)$ og $z(t)$ utvikler seg over et tidsintervall [0,T]:

$$\begin{aligned} x'(t) &= a(y(t) - x(t)) \\ y'(t) &= bx(t) - y(t) - x(t)z(t) \\ z'(t) &= x(t)y(t) - cz(t) \end{aligned}$$

Ved tiden t=0 har vi initialbetingelsene x(0)=x0, y(0)=y0 og z(0)=z0. Konstantene a, b, c er alle positive og i tillegg skal vi ha $a > c + 1$.

a) Skriv en funksjon **Lorenz(x0, y0, z0, a, b, c, T, N)** som tar initialverdier x0, y0 og z0, parametrene a, b og c, slutt-tidspunktet T, og antall løsningspunkter N som argumenter, og som beregner og returnerer løsningen for tidene $t_k = k \cdot \Delta$ hvor $k = 0, 1, \ldots, N-1$ og $\Delta = T/(N-1)$. Bruk RungeKutta4-metoden i modulen ODESolver for å løse differensiallikningene. De relevante delene av ODESolver ligger som vedlegg til denne oppgaven. Funksjonen **Lorenz** skal returnere fire numpy-arrayer, alle av lengde N:

t, som inneholder tidspunktene t_k hvor den numeriske løsningen er beregnet,
x, som inneholder $x(t_0), x(t_1), \ldots, x(t_{N-1})$
y, som inneholder $y(t_0), y(t_1), \ldots, y(t_{N-1})$
z, som inneholder $z(t_0), z(t_1), \ldots, z(t_{N-1})$.

b) Vi ønsker å løse modellen med initialbetingelsene $x_0 = 0$, $y_0 = 2$, $z_0 = 0$ og med parameterverdiene $a = 4$, $b = 16$, $c = 0.5$. Skriv kode for å kalle funksjonen **Lorenz** med de oppgitte parametrene og med T=100 og N=500. Skriv også kode for å plotte grafene til løsningene $x(t)$, $y(t)$ og $z(t)$. De tre grafene skal plottes i hver sin farge i samme vindu, og med en legend som forklarer hvilken farge som svarer til henholdsvis $x(t)$, $y(t)$ og $z(t)$. Ta med nødvendig import.

c) Det er kjent at Lorenz-modellen er veldig følsom for initialbetingelsene, dvs at selv veldig små endringer i verdiene til x(0), y(0) eller z(0) kan gi store utslag i verdiene til $x(t)$, $y(t)$ og $z(t)$ på et senere tidspunkt t. For å undersøke dette skal vi tenke oss at verdiene til parametrene a, b, c og de to siste initialbetingelsene y_0, z_0 holdes fast, slik at verdiene til $x(t)$, $y(t)$ og $z(t)$ på tidspunktet $t = T$ bare er bestemt av (dvs er en funksjon av) initialbetingelsen x_0. Bruk de samme verdiene for parametrene a, b, c og for y_0, z_0 som i deloppgave b). La også T og N ha de samme verdiene som vi brukte der. Skriv kode som velger 100 ulike verdier for initialbetingelsen x_0 på intervallet [0, 0.01] og for hver slik verdi løser Lorenz-modellen og tar vare på verdien til $x(t)$ på tidspunktet t=T. Skriv deretter kode for å plotte verdien til $x(T)$ som en funksjon av verdien til x_0.

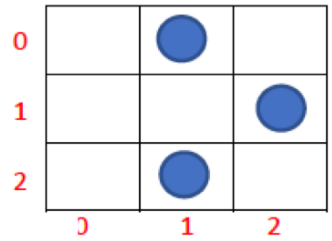
Skriv ditt svar her

```
1
```

Maks poeng: 9

14 Brettspill

Vi skal her programmere et enkelt brettspill. La N > 1 være et heltall. Vi har et brett som består av N x N felter hvor rader og kolonner er nummerert fra 0 til N-1. Vi har også N brikker plassert på brettet (aldri to i samme felt). Her er et eksempel hvor N = 3:



Brettets størrelse og brikkenes startposisjon ligger i tekstfilen start.txt. Første linje angir N og de neste N linjene angir brikketype, radnummer og kolonnenummer (separert med komma) for hver brikke. Det finnes to ulike typer brikker og disse står angitt som 1 eller 2 i filen. Slik ville filen sett ut for eksemplet ovenfor:

```
3
2,0,1
1,1,2
1,2,1
```

Her er første brikke av type 2, mens de to neste er av type 1.

Vi skal implementere spillet som en klasse:

```
class Game:
  def __init__(self):
    # Her skal det stå kode (se deloppgave a under)

  def move1(self, row0, col0, row1, col1):
    # Her skal det stå kode (se deloppgave b under)

  def move2(self, row0, col0, row1, col1):
    # Her skal det stå kode (se deloppgave c under)
```

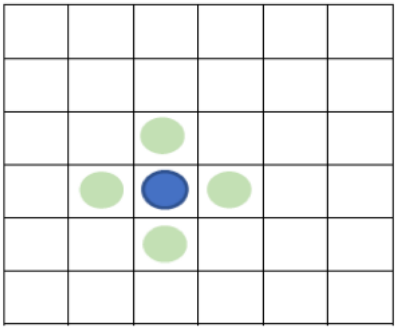
a) Skriv ferdig konstruktøren i klassen Spill som skal lese filen start.txt og lagre brikkenes posisjoner i en 2-dimensjonal numpy-array **board** med N rader og N kolonner. Denne variabelen skal være en instansvariabel og kan opprettes slik (når N er lest fra fil):

```
import numpy as np
self.board = np.zeros((N,N), 'int')
```

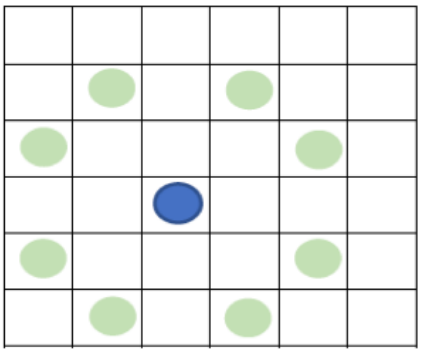
Når du har lagret brikkeposisjonene, skal self.board[i,j] == 0 hvis det ikke står noen brikke i feltet i posisjon i'te rad og j'te kolonne, self.board[i,j] == 1 hvis det står en brikke av type 1 i feltet, og self.board[i,j] == 2 hvis det står en brikke av type 2 i feltet.

b) Skriv ferdig funksjonen **move1(self, row0, col0, row1, col1)** som flytter en brikke av type 1 fra posisjonen row0, col0 til posisjonen row1, col1 ved å gjøre nødvendige endringer i

instansvariabelen **board**. Først må funksjonen sjekke at flyttet er lovlig. Kravene for dette er at (a) det må allerede stå en brikke av riktig type i posisjon row0, col0 og det nye feltet row1, col1 må stå ledig; (b) brikker kan bare flytte ett felt horisontalt (venstre eller høyre) eller ett felt vertikalt (nedover eller oppover); og (c) brikkene må ikke flyttes til posisjoner utenfor brettet. Figuren under viser et eksempel med alle lovlige flytt for en slik brikke. Her er N=6, blå angir startposisjon, og grønn angir mulige sluttposisjoner. Hvis flyttet ikke er lovlig skal funksjonen avslutte uten å endre variabelen **board**, men det er ikke nødvendig å skrive ut en feilmelding eller beskjed om at flyttet er ulovlig.



c) Skriv ferdig funksjonen **move2(self, row0, col0, row1, col1)** som flytter en brikke av type 2 fra posisjonen row0, col0 til posisjonen row1, col1 ved å gjøre nødvendige endringer i instansvariabelen **board.** Også her må du først sjekke at flyttet er lovlig. Kravene for dette er at det må stå ei brikke av riktig type i feltet row0, col0, og det nye feltet row1, col1 må stå ledig. Brikker av type 2 beveger seg alltid to felter vertikalt og ett felt horisontalt, eller to felter horisontalt og ett felt vertikalt. I tillegg må du sjekke at flyttet ikke fører til at en brikke havner utenfor brettet. Figuren under viser et eksempel med alle lovlige flytt for en slik brikke. Her er N=6, blå angir startposisjon, og grønn angir mulige sluttposisjoner. Hvis flyttet ikke er lovlig skal funksjonen avslutte uten å endre variabelen **board**, men det er ikke nødvendig å skrive ut en feilmelding eller beskjed om at flyttet er ulovlig.



Skriv ditt svar her

```
1
```

Maks poeng: 9

15 B-splines

B-splines er en type funksjoner som brukes blant annet til datagrafikk og numerisk derivasjon. I denne oppgaven skal du lage funksjoner i Python som beregner verdiene til slike B-splines.

Utgangspunktet for B-splines er et sett av verdier (heretter kalt knutepunkter) $x_{-m} < x_{-m+1} < \ldots < x_{n+m}$ hvor $m > 0$ og $n > 0$ (legg merke til at de første m elementene har negativ indeks). *B-splines funksjonene av grad 0* kalles $B^0_{-m}(t), B^0_{-m+1}(t), \ldots, B^0_{n+m-1}(t)$ og er definert som følger:

$$B^0_p(t) = \begin{cases} 0, & t < x_p \text{ eller } t > x_{p+1} \\ (x_{p+1} - x_p)^{-1}, & x_p < t < x_{p+1} \\ \frac{1}{2}(x_{p+1} - x_p)^{-1}, & t = x_p \text{ eller } t = x_{p+1} \end{cases}$$

hvor $p = -m, -m+1, -m+2, \ldots, n+m-1$ og hvor t er et reelt tall.

a) Lag en Python-funksjon **Bspline0(t, p, x)** som beregner og returnerer verdien $B^0_p(t)$. Argumentene til funksjonen er t (en float-verdi), p (et heltall), og x (en dictionary med alle knutepunktene, angitt slik at verdiene $x_{-m}, x_{-m+1}, \ldots, x_{n+m}$ kan hentes ut som x[-m], x[-m+1], ..., x[n+m]). Du kan lett finne verdiene til *m* og *n* ved å se på laveste og høyeste nøkkelverdi i x (disse er henholdsvis lik *-m* og *n+m*). Du kan anta at p ligger i riktig intervall (dvs at p har en av verdiene $-m, -m+1, -m+2, \ldots, n+m-1$) .

B-splines funksjonene av grad $k > 0$ kalles $B^k_{-m}(t), B^k_{-m+1}(t), \ldots, B^k_{n+m-k-1}(t)$ og disse er definert som følger:

$$B^k_p(t) = \frac{(t - x_p)B^{k-1}_p(t) + (x_{p+k+1} - t)B^{k-1}_{p+1}(t)}{x_{p+k+1} - x_p}$$

hvor $p = -m, -m+1, -m+2, \ldots, n+m-k-1$.

b) Lag en Python-funksjon **Bspline(t, k, p, x)** som beregner og returnerer verdien $B^k_p(t)$. Argumentene til funksjonen er t (en float-verdi), k (et ikkenegativt heltall), p (et heltall), og x (en dictionary med alle knutepunktene, angitt slik at verdiene $x_{-m}, x_{-m+1}, \ldots, x_{n+m}$ kan hentes ut som x[-m], x[-m+1], ..., x[n+m]). Du kan også her anta at p ligger i riktig intervall (dvs at p har en av verdiene $-m, -m+1, -m+2, \ldots, n+m-k-1$).

Skriv ditt svar her

```
1
```

Maks poeng: 6

16 Skal ikke besvares, til bruk under sensuren

Denne oppgaven skal ikke besvares, men blir brukt av sensor for å legge inn poeng fra midtveis-eksamen.

Skal ikke besvares.

Format

Words: 0

Maks poeng: 25

Question 13
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
```

Question 6
Attached

```
# Alternative A
y = -1
for e in x:
   if e > y:
      y = e

# Alternative B
y = x[-1]
for i in range(len(x)):
   y = max(y,x[i])

# Alternative C
for i in range(len(x)-1):
   if x[i+1] < x[i]:
      x[i+1] = x[i]
y = x[-1]

# Alternative D
v = [x[i] for i in range(1, len(x)) if x[i] > x[i-1]]
y = v[-1]

# Alternative E

import numpy as np
v = np.asarray(x)
y = -min(-v)

# Alternative F
v = [x[0]]
for i in range(len(x)):
   if v[-1] <= x[i]:
      v.append(x[i])
y = v[-1]

# Alternative G
s = f"y=max({x})"
exec(s)
```