Erfaringer med digital eksamen fra TDT4127 – Programmering og Numerikk

Torbjørn Ringholm

Bakteppe

Høsten 2018 ble faget TDT4127 – Programmering og Numerikk – undervist for første gang for rundt 150-200 elever. Dette er et midlertidig fag som undervises i en periode inntil alle landets ingeniørutdanninger oppfyller FUS sitt krav om IKT-innhold for sivilingeniørgraden. Målgruppen for faget er masterstudenter ved tekniske fag som tidligere har hatt mindre enn 5 stp med programmering i sine tidligere bachelorstudier. Faget ble undervist i en hybridløsning med TDT4110 – IT Grunnkurs – i en ordning der studentene ukentlig hadde 2 timer forelesning sammen med studentene i ITGK for programmeringsferdigheter i Python og 1 time forelesning med numerikk, med 2 timer øvingsforelesning og omfattende IT-lab-ordning for personlig hjelp.

Numerikkpensumet erstatter den ordinære IT-teorien fra ITGK. Det var tiltenkt mest som et «praksiskurs» med mål om å opplyse studentene om ulike metoder innen numerikk, og vektla implementasjon av metoder gjennomgått i den ukentlige numerikktimen med et minimum av teoretisk bakgrunn. I alt var arbeidsfordelingen i faget anslått til 1/3 numerikk, 2/3 programmering på forhånd. Fagplan finnes i appendiks.

Eksamenen var tiltenkt å hovedsakelig teste programmeringskunnskaper, med en viss andel numerikkteori dekt av multiple choice-oppgaver. Samtidig valgte TDT4110 å gå over til digital eksamen gjennom plattformen Inspera, og det var da naturlig at TDT4127 også gjorde dét. Dette ga mulighet til å teste ut forskjellige oppgaveformer i Inspera.

Eksamen

Eksamen ble utformet sammen med Guttorm Sindre (IDI) og omfattet flere oppgavetyper. Foruten standard «frihånds» programmeringsoppgaver (oppgaver der kandidaten starter med et blankt tekstfelt og skal skrive en kode som oppfyller visse betingelser) prøvde vi ut en rekke oppgavetyper som per i dag har mer eller mindre innebygd støtte i Inspera. Disse er beskrevet i detalj i neste seksjon. For å forberede kandidatene på bruk av Inspera og på oppgavetypene var det satt opp to auditorieøvinger i løpet av semesteret som ble gjennomført i sin helhet på Inspera. Eksamen ble vurdert i ettertid av oss og av studentene gjennom et ekstra referansegruppemøte på nyåret.

Referansegruppen var fornøyde med ordningen med digital eksamen og hadde opplevd gjennomføringen som uproblematisk. De likte at det var en variasjon i oppgavetyper og følte eksamen hadde testet ferdighetene som var opparbeidet gjennom semesteret, men syntes mengden (20 deloppgaver) var litt i overkant. De var fornøyde med ordningen med auditorieøvinger for å forberede dem på eksamensformen.

Vår erfaring med eksamen var at det er mer krevende å utforme oppgaver i Inspera på grunn av brukervennlighet, og at tidsbesparende besvarelsesformer (kode er lettere å skrive og redigere på PC, samtidig som spørsmål med forhåndsdefinerte svar slik som flervalgsspørsmål) gjør det nødvendig å lage en større mengde spørsmål per eksamen. Dette kan igjen ha en effekt på kvaliteten på oppgavene da det er større sjanse for å gjøre feil i utformingen pga volumet. Etter sensur så vi at rundt 16 deloppgaver ville vært bedre enn de 20 vi gav, med tanke på gjennomføringsrate fra kandidatene. Vi opplevde også at selv om utformingen tar lengre tid så tar det kortere tid å sensurere oppgavene – spesielt på grunn av at man kan rette på oppgavebasis heller enn kandidatbasis, noe som bidrar til at sensuren blir mer robust mot skjevhet grunnet dagsform. Vi savnet imidlertid bedre verktøy for å eksportere poengdata fra Inspera til Excel – per i dag kan man kun hente ut endelig bokstavkarakter per kandidat, selv om det burde være enkelt å skaffe detaljert data om poengfordeling per deloppgave og kandidat i et digitalt system.

Det å gjennomføre auditorieøvinger i Inspera på forhånd var også en arbeidskrevende affære da Inspera i utgangspunktet ikke er egnet til noe annet enn eksamenssettinger. Det tok ekstra innsats både fra vit.ass. og fra Inspera-teamet å få stablet dette på beina, samtidig som det å utforme oppgavene til auditorieøvingene var arbeidskrevende. Når dét var på plass var det greit å gjennomføre øvingene. Vi fikk gode tilbakemeldinger fra studentene om hva som fungerte og ikke fungerte og som påvirket utformingen av eksamen, så denne ordningen var nyttig i læringsprosessen for både oss og kandidatene.

Under selve eksamen fungerte selve Inspera-systemet fint, og vi opplevde gjennomføringen som problemfri. Man bør imidlertid være obs på at endringer som gjøres på eksamensoppgavene i de siste dagene før eksamen ikke nødvendigvis blir registrerte – dette må man kontakte Inspera-teamet for å få sikkerhet i. Vi opplevde at små skrivefeilskorreksjoner som ble gjort noen kvelder i forveien øyensynlig ble registrerte på Inspera-plattformen, men på selve dagen var det en eldre versjon av eksamen som ble tilgjengelig for studentene. Inspera tillater imidlertid å sende ut en-til-alle-meldinger til studentene med informasjon om korreksjon. Disse er tilgjengelige i brukergrensesnittet og gjør at man slipper å rope ut korreksjonen i salen. Dessverre var det ingen signalement i brukergrensesnittet som informerte om denne meldingen, så vi møtte likevel gå rundt og informere om at det var kommet en beskjed om korrektur. Korrekturordningen hadde for øvrig et veldig begrenset format som gjorde beskjeden ikke kunne inneholde linjeskift eller avsnitt – dette gjorde korrekturbeskjeden tunglest for studentene.

Oppgavetyper

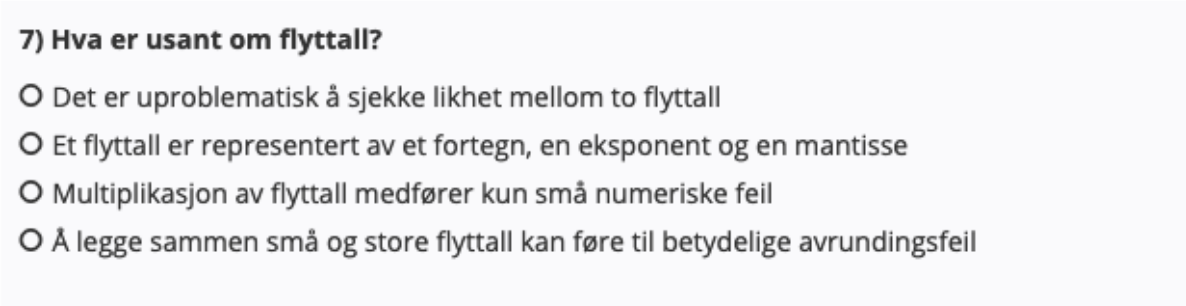
Foruten standard langsvarsoppgaver prøvde vi ut flere av oppgavetypene som støttes i Inspera. Noen av disse er automatisk rettet, som gjør at man sparer seg for en del arbeid i sensurprosessen. Oppgavetypene beskrives under, med en eksempeloppgave per type og min vurdering av hvorvidt oppgavetypen egner seg for matematiske oppgaver.

Flervalgsspørsmål

ITGK, som TDT4127 bygger på, har hatt tradisjon for å ha en flervalgsdel på eksamen for å teste studentenes teorikunnskaper. Vi valgte å videreføre dette og opplevde at vi fikk dekt de fleste teoretiske temaene i pensum på en rimelig måte samtidig som vi kunne regne med at teoribiten gikk fort nok til at studentene hadde god tid til å svare på de mer praktiske programmeringsoppgavene.

Studentene opplevde spørsmålene som rimelige, og vår statistikk tilsa at oppgavene ble krevende nok til at man kunne differensiere mellom kandidatene. Vi valgte å ikke gi minuspoeng for feil besvarte oppgaver da en slik ordning kan føre til at mindre selvsikre studenter svarer blankt når de tviler.

Hvordan kan man bruke dette i et matematisk fag?
Jeg ser for meg at dersom man skal teste rene faktakunnskaper så er dette et rimelig verktøy, eller dersom man vil teste enkle resonnement gjennom å finne nøyaktige tallsvar.


7) Hva er usant om flyttall?

- ○ Det er uproblematisk å sjekke likhet mellom to flyttall
- ○ Et flyttall er representert av et fortegn, en eksponent og en mantisse
- ○ Multiplikasjon av flyttall medfører kun små numeriske feil
- ○ Å legge sammen små og store flyttall kan føre til betydelige avrundingsfeil

Dra og slipp

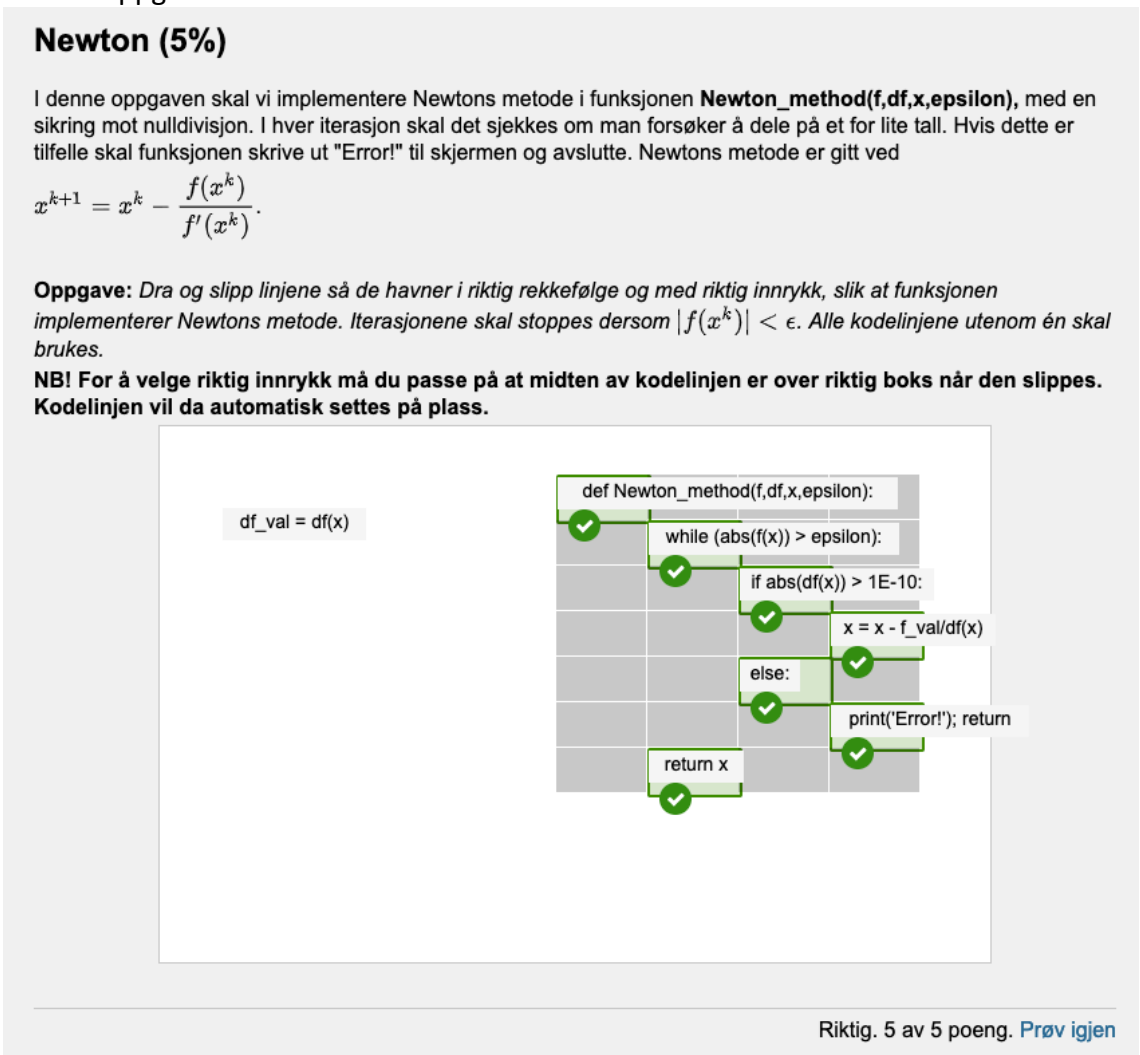
Dette er en ny type oppgave som ble muliggjort med Inspera. Studentene skulle dra kodelinjer over i et rutenett som representerte ulike linjer og innrykk i en Python-IDE slik at koden fikk riktig flyt og ble syntaktisk korrekt. Dette er en uortodoks bruk av Inseras dra-og-slipp-oppgaver som krevde en del ekstraarbeid for å få til å fungere. Blant annet må rutenettet av bokser settes opp manuelt, noe som var arbeidskrevende (et par timer første gang!) i Insperas grensesnitt som kan oppfattes som frustrerende.

Studentene fikk prøve denne oppgaveformen på auditorieøvinger. Den fungerte godt i praksis, men den automatiske rettingen bør kontrolleres i etterkant, spesielt om man ikke ønsker å straffe følgefeil av typen «Linje 2 har feil innrykk, følgelig har samtlige linjer feil innrykk og premieres ikke».

Hvordan kan man bruke dette i et matematisk fag?

Oppgaveformen er egnet for å teste kandidatens forståelse av programflyt og Python-syntaks, men jeg har vanskelig for å se hvordan den kan brukes til å teste matematisk forståelse. Det er nærliggende å tenke på oppgaver der man skal stokke argumenter i et bevis i riktig rekkefølge, men det setter ikke noe annet enn den logiske sansen på prøve.

Slik ser oppgavene ut for studentene:



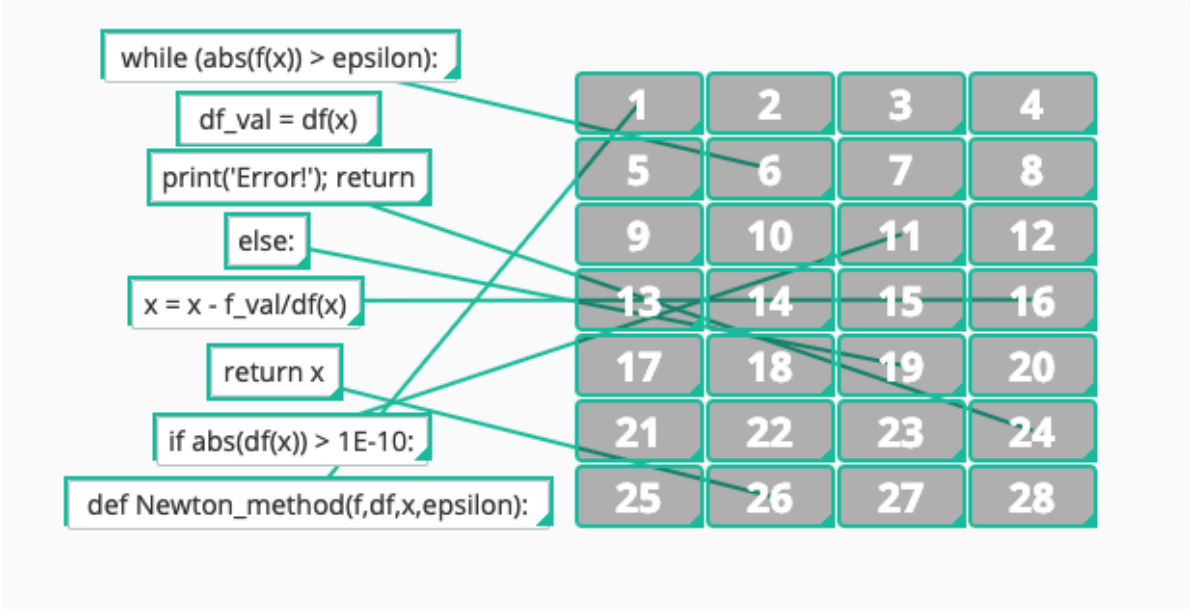
Slik ser oppgaven ut i Insperas visning for faglærer:

I denne oppgaven skal vi implementere Newtons metode i funksjonen **Newton_method(f,df,x,epsilon),** med en sikring mot nulldivisjon. I hver iterasjon skal det sjekkes om man forsøker å dele på et for lite tall. Hvis dette er tilfelle skal funksjonen skrive ut "Error!" til skjermen og avslutte. Newtons metode er gitt ved

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}.$$

Oppgave: *Dra og slipp linjene så de havner i riktig rekkefølge og med riktig innrykk, slik at funksjonen implementerer Newtons metode. Iterasjonene skal stoppes dersom* $|f(x^k)| < \epsilon$. *Alle kodelinjene utenom én skal brukes.*

NB! For å velge riktig innrykk må du passe på at midten av kodelinjen er over riktig boks når den slippes. Kodelinjen vil da automatisk settes på plass.



Utfylling

En litt mer utfordrende variant av nedtrekksoppgavene – her skal kandidaten fylle ut manglende kode slik at metoden implementeres på riktig måte.

Oppgavetypen sjekker kandidatenes kodeforståelse, evne til å oversette matematikk til kode, og grep om Python-syntaks.

Per nå er oppgavene ikke automatisk rettet – det finnes støtte for dette i Inspera, men vi vurderte det dithen at Inspera ikke er i stand til å skille mellom skrivefeil og forståelsesfeil og at man derfor bør manuelt rette oppgavene.

Hvordan kan man bruke dette i et matematisk fag?

I noen numerikk/statistikk-fag kan man ønske å sette kandidatens programmeringsegenskaper på prøve dersom det ikke allerede er tilstrekkelig dekket gjennom prosjektarbeid. I så fall kunne det tenkes at en utfyllingsoppgave kan være en lettere deloppgave i en slik vurdering, for eksempel ved at rammeverket rundt en algoritme er satt opp og at det er opp til kandidaten å implementere selve algoritmen.

I denne oppgaven skal du fullføre en implementasjon av eksplisitt Euler. Funksjonen **explicit_Euler_list(F,x,N,T)** skal ta inn en funksjon **F**, et startpunkt **x**, antall tidssteg **N** og stoppetidspunkt **T** og returnere en liste med alle iterasjonene til eksplisitt Euler-metoden som forklart nedenfor, *inkludert startpunktet*.

Eksplisitt Euler-metoden er gitt ved iterasjonene

$$x^{k+1} = x^k + hF(x^k, t^k), \qquad t^k = kh$$

Oppgave: *Implementér funksjonen explicit_Euler_list(F,x,N,T). Du skal ta utgangspunkt i koden under og fylle ut feltene merket "###".*

```
def explicit_Euler_list(F,x,N,T):
    h = T/N
    x_list = [###]
 for k in range(N):
        t_k = ###
        x = ###
        x_list.###
 return ###
```

Skriv ditt svar her...

```
function add(x, y) {
    var resultString = "Hello";
    var result = x + y;
    return resultString + result;
}

var addResult = add(3, 2);
console.log(addResult);
```

Nedtrekk

Dette er en enklere form for programmeringsoppgave i flervalgsmodus som sjekker kandidatenes kodeforståelse, evne til å oversette matematikk til kode, og grep om Python-syntaks.

Oppgavene kan rettes automatisk uten problem.

Hvordan kan man bruke dette i et matematisk fag?
På samme måte som utfyllingsoppgavene kan dette inngå som en enkel deloppgave som vurderer kandidatens forståelse av implementasjon av algoritmer.

I denne oppgaven skal du implementere Simpsons kompositttmetode, som er gitt ved

$$\int_a^b x(t)\mathrm{d}t \approx \frac{h}{3}\left(x_0 + 4x_1 + 2x_2 + \ldots + 2x_{N-2} + 4x_{N-1} + x_N\right),$$

$$x_k = x(t_k), \qquad t_k = kh \qquad h = \frac{b-a}{N}$$

Funksjonen **comp_simpson_method(x_list,a,b)** skal ta inn en liste **x_list** som inneholder x_k, for $k = 0,\ldots,N$, samt et integrasjonsintervall [**a, b**].

Oppgave: *Velg riktige alternativer fra menyene slik at* **comp_simpson_method(x_list,a,b)** *implementerer Simpsons metode der x_list er en liste* $[x_0, x_1, \ldots, x_N]$ *som inneholder punktene som skal brukes i formelen for Simpsons metode, og a og b er venstre og høyre endepunkt i integralet.*

```
def comp_simpson_method(x_list,a,b):
    N = [len(x_list)-1]
    h = (b-a)/float(N)
    total_sum [= x_list[0]]
  for j in range(1,N):
    if [j % 2 is 0:]
            total_sum += 2*x_list[j]
    else:
            [total_sum += 4*x_list[j]]
    total_sum += x_list[N]
    total_sum = [h/3*total_sum]
  return total_sum
```

Feilsøking

Dette er en oppgavetype som tester kandidatens kodeforståelse. Kandidaten får oppgitt en eller flere kodesnutt(er) som inneholder en eller flere feil. De bes om å finne feilen og beskrive hva den går ut på.

Oppgavene er ikke automatisk rettet og besvares i fritekst.

Hvordan kan man bruke dette i et matematisk fag?
Det blir kanskje litt søkt, men man kan se for seg at en kandidat presenteres med et feilaktig eller mangelfullt bevis og bes om å rette opp i eller greie ut om detaljer som er feil/mangelfulle.

Funksjonene under svarer til operasjoner som er kreves i Gauss-eliminasjon og har **én feil hver**.

- **add(row1,row2,num)** skal ta inn lister **row1** og **row2** av samme lengde og legge **num** ganger **row1** til **row2**.
- **swap(row1,row2)** skal ta inn lister **row1** og **row2** av samme lengde og bytte om på elementene i de to listene.
- **get_max_row(A,row,col)** skal ta inn en n×n-matrise **A** og to indekser **row** og **col**. Funksjonen skal returnere radindeksen i kolonne nummer **col** der maksimal absoluttverdi av **A** finnes, når man starter fra rad nummer **row**.

Oppgave: *For hver funksjon under, finn kodefeilen. Beskriv de tre feilene i tekstfeltet nedenfor.*

```
def add(row1, row2, num):
  for i in range(0,len(row2)):
        row2[i] = row1[i] + num*row2[i]

def swap(row1,row2):
  for i in range(0,len(row2)):
        temp = row2[i]
        row1[i] = temp
        row2[i] = row1[i]

def get_max_row(A,row,col):
    currentMax = abs(A[row][col])
    currentMaxIndex = row
  for i in range(row+1,len(A)):
   if abs(A[i][col]) > currentMax:
          currentMax = abs(A[col][i])
          currentMaxIndex = i
  return currentMaxIndex
```

Matching

Dette er en måte å teste kodeforståelse på ved å knytte sammen input med output. Kandidaten får oppgitt en kode, og skal vise forståelse for hva den gjør ved å koble forskjellige inputparametere med korrekte outputverdier i et rutenett.

Oppgavene kan rettes automatisk.

Hvordan kan man bruke dette i et matematisk fag?
Dette er en ganske allsidig oppgaveform som man godt kan finne bruksområder for, nok en gang i «plankeoppgave»-form. For eksempel kan man på forhånd definere funksjoner f, g og h, og i oppgaven be kandidaten koble mutasjoner av disse via (for eksempel multiplikasjoner, komposisjoner, konvolusjoner, fouriertransformer og lignende) med riktige uttrykk.

Vi har definert tre globale variable:

- A = {0,1,2,3}
- B = {3,4,5}
- C = set(range(8))

Vi har dessuten funksjonen **count_list(lst)** gitt som følger:

```
def count_list(lst):
    result = [0]*5
    for s in lst:
        for i in range(1,6):
            if i in s:
                result[i-1] += 1
    return result
```

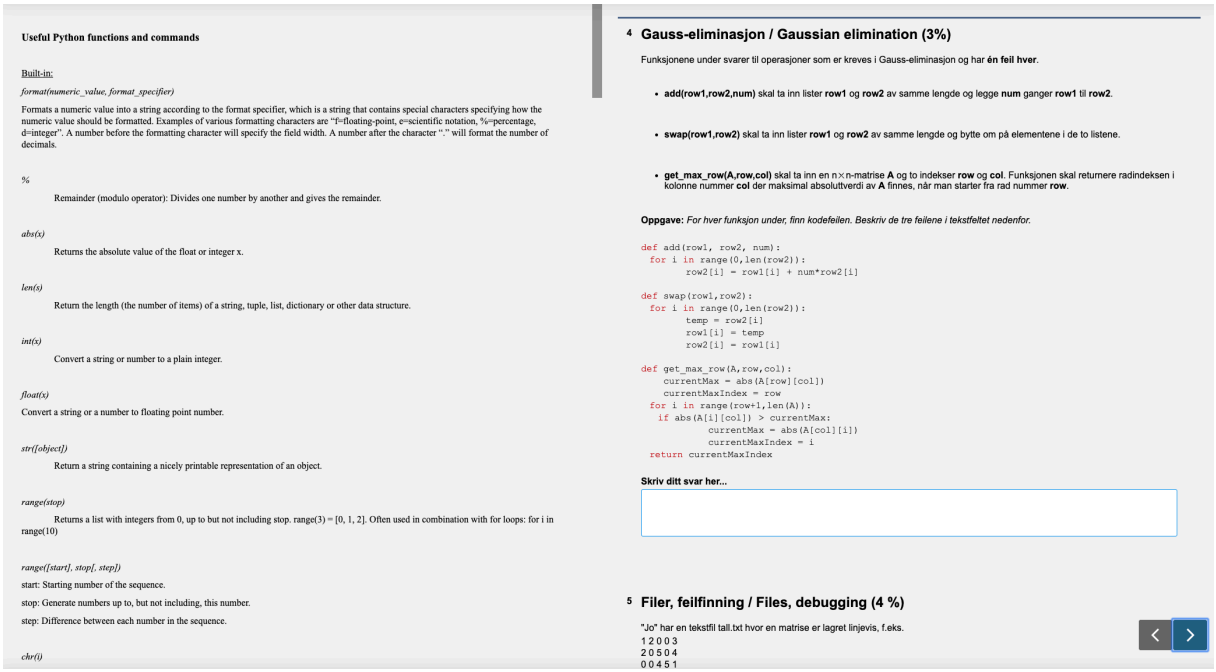
Hver rad-tittel i tabellen er et mulig argument til funksjonen count_list(). Hver kolonneoverskrift er en mulig returverdi. Marker hvilket argument som fører til hvilken returverdi.

	[1,1,1,0,0]	[1,1,1,1,1]	[0,0,0,0,0]	[2,2,3,2,2]	[6,6,6,6,6]
[A]	◉	○	○	○	○
[A.union(B)]	○	◉	○	○	○
[A.difference(C)]	○	○	◉	○	○
[A,B,C]	○	○	○	◉	○
[C]*3+[C]*3	○	○	○	○	◉

Vedlegg

Det er mulighet for å lage vedlegg i Inspera, og disse kan enten vises som en egen side i oppgaven eller i en side-ved-side-visning sammen med oppgaveteksten (zoom inn på bilde under for et eksempel).

Vår erfaring er at side-ved-side-visningen ble best mottatt av studentene da de gjerne ønsket å kunne se oppgaveteksten og vedlegget samtidig av åpenbare grunner. Eksamenskontoret var fast bestemt på at de ikke ville trykke vedlegget og dele ut under eksamen, så vi endte på side-ved-side-visningen. Baksiden er at hos noen studenter med liten PC-skjerm kan formateringen av oppgaveteksten bli dårlig pga liten tekstbredde. Vi har gitt Inspera tilbakemelding på at man bør kunne minimere vedlegget (ved for eksempel å trykke på en venstreorientert pil) for å kunne se oppgaveteksten i fullskjerm.



Useful Python functions and commands

Built-in:

format(numeric_value, format_specifier)

Formats a numeric value into a string according to the format specifier, which is a string that contains special characters specifying how the numeric value should be formatted. Examples of various formatting characters are "f=floating-point, e=scientific notation, %=percentage, d=integer". A number before the formatting character will specify the field width. A number after the character "." will format the number of decimals.

%

Remainder (modulo operator): Divides one number by another and gives the remainder.

abs(x)

Returns the absolute value of the float or integer x.

len(s)

Return the length (the number of items) of a string, tuple, list, dictionary or other data structure.

int(x)

Convert a string or number to a plain integer.

float(x)

Convert a string or a number to floating point number.

str([object])

Return a string containing a nicely printable representation of an object.

range(stop)

Returns a list with integers from 0, up to but not including stop. range(3) = [0, 1, 2]. Often used in combination with for loops: for i in range(10)

range([start], stop[, step])

start: Starting number of the sequence.

stop: Generate numbers up to, but not including, this number.

step: Difference between each number in the sequence.

chr(i)

4 Gauss-eliminasjon / Gaussian elimination (3%)

Funksjonene under svarer til operasjoner som er kreves i Gauss-eliminasjon og har **én feil hver**.

- **add(row1,row2,num)** skal ta inn lister **row1** og **row2** av samme lengde og legge **num** ganger **row1** til **row2**.
- **swap(row1,row2)** skal ta inn lister **row1** og **row2** av samme lengde og bytte om på elementene i de to listene.
- **get_max_row(A,row,col)** skal ta inn en $n \times n$-matrise **A** og to indekser **row** og **col**. Funksjonen skal returnere radindeksen i kolonne nummer **col** der maksimal absoluttverdi av **A** finnes, når man starter fra rad nummer **row**.

Oppgave: *For hver funksjon under, finn kodefeilen. Beskriv de tre feilene i tekstfeltet nedenfor.*

```
def add(row1, row2, num):
  for i in range(0,len(row2)):
        row2[i] = row1[i] + num*row2[i]

def swap(row1,row2):
  for i in range(0,len(row2)):
        temp = row2[i]
        row1[i] = temp
        row2[i] = row1[i]

def get_max_row(A,row,col):
    currentMax = abs(A[row][col])
    currentMaxIndex = row
  for i in range(row+1,len(A)):
    if abs(A[i][col]) > currentMax:
          currentMax = abs(A[col][i])
          currentMaxIndex = i
  return currentMaxIndex
```

Skriv ditt svar her...

5 Filer, feilfinning / Files, debugging (4 %)

"Jo" har en tekstfil tall.txt hvor en matrise er lagret linjevis, f.eks.

1 2 0 0 3
2 0 5 0 4
0 0 4 5 1

Appendiks

Fagplan

Faglig innhold

Emnet består at to deler: Grunnleggende prosedyreorientert programmering i Python (2/3) og numerikk (1/3).

Prosedyreorientert programmering:

- Variabler og datatyper.
- Input og output.
- Kontrollstrukturer: Sekvens, valg, og repetisjon.
- Strukturering og oppdeling av programmer; funksjoner og moduler.
- Datastrukturer: Lister, tabeller, tekststrenger, mengder, tupler og poster (dictionary).
- Filbehandling, persistent lagring av informasjon, og unntak (exception).
- Rekursjon.
- Python som programmeringsomgivelse.
- Behandling av N-dimensjonale matriser
- Plotting av funksjoner

Numerikk:

- Numerisk integrasjon av funksjoner: Trapes-metoden, Simpsons metode, Adaptiv Simpson metode
- Newtons metode for å finne funksjoners nullpunkt
- Gauss-eliminasjon for å løse lineære ligningssett
- Numerisk løsning av ordinære differensialligninger

Læringsutbytte

Kunnskaper:

- Har grunnleggende kunnskap om grunnelementene i prosedyreorientert programmering.
- Har grunnleggende kunnskap om prosessen fra problem til fungerende program.
- Har grunnleggende kunnskap om innen numeriske metoder.

Ferdigheter:

- Kan anvende grunnelementene i praktisk, prosedyreorientert programmering.
- Kan bruke relevante programmeringsverktøy.
- Kan analysere et problem, finne prosess og datastrukturer som løser problemet, formulere en løsning som pseudokode eller flytskjema, og programmere og teste en løsning i Python.
- Kan gjennomføre mindre programmeringsprosjekter.
- Kan utføre numeriske beregninger i Python.