

Eksamen vinter 2025

Kursusnavn:	Programmering
Kursusnummer:	02002 og 02003
Eksamensdato:	21. december 2025
Tilladt hjælpemidler:	Alle hjælpemidler, intet internet
Eksamenens varighed:	4 timer
Vægtning:	Alle opgaver har samme vægt
Antal opgaver:	10
Antal sider:	12

Contents

- Eksamensinstruktioner
- Opgave 1: Kaliberdiameter (Gauge Diameter)
- Opgave 2: Gateafstand (Gate Distance)
- Opgave 3: Parallelle modstande (Parallel Resistors)
- Opgave 4: Graf-transponering (Graph Transpose)
- Opgave 5: Ulige gennemsnit (Odd Means)
- Opgave 6: Navigationssporing (Navigation Tracking)
- Opgave 7: Sætningskompleksitet (Sentence Complexities)
- Opgave 8: Cykellygte (Bike Light)
- Opgave 9: Kuffert (Suitcase)
- Opgave 10: Butiksstatus (Shop Status)

Eksamensinstruktioner

Eksamensmateriale

Eksamensmaterialet er en enkelt zip-fil, som du skal udpakke til en mappe på din computer. Zip-filen indeholder eksamensopgaven som et pdf-dokument på engelsk `2025_12_exam_English.pdf` og det samme dokument på dansk `2025_12_exam_Danish.pdf` (dette dokument). Hver pdf indeholder alt, hvad der er nødvendigt for at løse eksamen.

Som ekstra hjælp indeholder zip-filen en mappe `2025_12_exam` med følgende indhold:

- Tomme Python-filer `<task_name>.py`, hvor `<task_name>` er navnet på opgaven. Der kan være færre filer end opgaver.
- En Python-testfil for hver opgave, `test_task_<n>_<task_name>.py`, hvor `<n>` er opgavens nummer, og `<task_name>` er navnet på opgaven.
- En Python-fil `test_tasks_all.py`.
- En mappe `files`, der indeholder datafiler, hvis der er nogen.

Løsning af eksamensopgaver

For at kunne løse eksamensopgaverne skal du have en computer med Python installeret. Alle opgaver kan løses ved hjælp af en editor efter eget valg, f.eks. IDLE eller VS Code.

Vi anbefaler, at du bruger de medfølgende filer: Skriv dine løsninger i de tomme filer, og test dem ved at køre de medfølgende test-scripts. Disse scripts kontrollerer, om din løsning fungerer korrekt for inputtet i opgavebeskrivelsen. Brug

`test_tasks_all.py` til at køre alle test på én gang. De medfølgende test-scripts antager, at *current working directory* er `2025_12_exam`. Hvis du bruger VS Code, skal du derfor klikke på `File` → `Open Folder...` og vælge mappen `2025_12_exam` (som findes *inde* i den mappe, du har udpakket).

En løsning, der får fejl i den medfølgende test, er forkert. Hvis din løsning består den medfølgende test, er det ikke en garanti for, at din løsning er korrekt.

Alle opgaver kan løses ved hjælp af de værktøjer, der undervises i på kurset. Løsninger, der importerer andre moduler end `math`, `numpy`, `os` eller `matplotlib`, vil ikke blive bedømt. De udleverede test-scripts kontrollerer ikke dette, så det er dit ansvar at sikre, at dine løsninger kun bruger de tilladte moduler.

Hvis du mener, at der er en fejl eller uklarhed i teksten, skal du bruge den mest rimelige fortolkning af teksten og løse opgaven efter bedste evne. Hvis vi efter eksamen finder uoverensstemmelser i en eller flere opgaver, vil dette blive taget i betragtning under bedømmelsen.

Evaluerings af eksamen

Vi vil køre yderligere tests for hver opgave for at kontrollere, om dine løsninger fungerer som angivet i opgaven. Andelen af korrekte tests er scoren for hver opgave. Den samlede score er gennemsnittet af scoren for hver opgave.

Aflevering

For at aflevere dine løsninger skal du uploade dine Python-filer med løsningerne til Digital Exam-systemet. I Digital Exam-systemet kan filer afleveres enten som *hoveddokument* eller som *bilag*. Du kan uploade *enhver* af dine løsningsfiler som hoveddokument og de resterende filer som bilag.

Indsend kun følgende filer med præcis disse navne. Alle andre filer vil blive ignoreret.

- `bike_light.py`
- `gate_distance.py`
- `gauge_diameter.py`
- `graph_transpose.py`
- `navigation_tracking.py`
- `odd_means.py`
- `parallel_resistors.py`
- `sentence_complexities.py`
- `shop_status.py`
- `suitcase.py`

Opgave 1: Kaliberdiameter (Gauge Diameter)

En 12-kaliber kugle fremstilles ved at tage et pund bly, dele det i tolv lige store dele og rulle hver del til en kugle. For en anden kaliber, f.eks. 8-kaliber, anvendes samme procedure, men med otte lige store dele.

Derfor opfylder diameteren d af en n -kaliber kugle

$$\frac{\pi}{6} d^3 = \frac{V}{n},$$

hvor V er volumenet af et pund bly, som er 40 cm^3 .

Skriv en funktion, der, givet n , returnerer kuglens diameter i centimeter.

For eksempel for en kugle med kaliber 12:

$$d^3 = \frac{6V}{n\pi} = \frac{6 \cdot 40 \text{ cm}^3}{12\pi} = 6.3662 \text{ cm}^3,$$

$$d = \sqrt[3]{6.3662 \text{ cm}^3} = (6.3662 \text{ cm}^3)^{\frac{1}{3}} = 1.8534 \text{ cm}.$$

Diameteren på kuglen med kaliber 12 er altså 1.8534 cm , og funktionen bør opføre sig som vist nedenfor.

```
>>> gauge_diameter(12)
1.8533610896304256
```

Filnavnet og kravene er:

gauge_diameter.py

`gauge_diameter(n)`

Computes the diameter of an n -gauge ball.

Parameters:

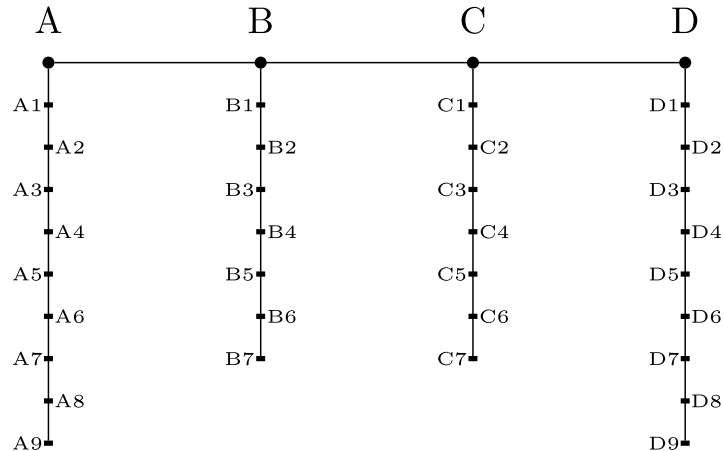
- `n` `int` The gauge.

Returns:

- `float` The diameter in cm.

Opgave 2: Gateafstand (Gate Distance)

En lufthavn har fire terminaler. Hver terminal har flere gates, som vist nedenfor.



I illustrationen er starten på hver terminal markeret med en cirkel. Det tager 4 minutter at gå fra starten af en terminal til starten af den næste terminal. Langs en terminal tager det 0.8 minutter at gå fra starten til den første gate og 0.8 minutter mellem hvert par nabogates. Vi skal beregne den hurtigste gangtid mellem to gates og oprette en besked, hvor den hurtigste gangtid rundes op til nærmeste hele minut.

Tag for eksempel gates A8 og B5. Det tager 8 gange 0.8 minutter at gå fra A8 til starten af terminal A. Det tager 4 minutter at gå fra starten af terminal A til starten af terminal B. Endelig tager det 5 gange 0.8 minutter at gå fra starten af terminal B til B5. Den hurtigste gangtid er derfor $4 + (8 + 5) 0.8 = 14.4$ minutter, hvilket rundes op til 15 minutter.

Skriv en funktion, der tager to gates som input. Gates angives som strenge, der indeholder et bogstav og et heltal. Funktionen skal returnere strengen `'walking time <n> min'` hvor `<n>` er den hurtigste gåtid mellem de to gates, rundet op til nærmeste hele minut. Funktionen skal også fungere korrekt, hvis lufthavnen udvides med flere gates. Du kan dog antage, at der kun vil være de fire terminaler A, B, C og D.

Funktionens opførsel er vist nedenfor.

```
>>> gate_distance('A8', 'B5')
'walking time 15 min'
```

Filnavnet og kravene er:

gate_distance.py

```
gate_distance(gate1, gate2)
```

Compute gate distance for two airport gates.

Parameters:

- `gate1` `str` Gate identifier containing a letter (terminal) and gate number.
- `gate2` `str` Gate identifier containing a letter (terminal) and gate number.

Returns:

- `str` Information about the walking distance between the gates.

Opgave 3: Parallele modstande (Parallel Resistors)

Når modstande er forbundet parallelt, kan den samlede modstand R beregnes ved hjælp af formlen:

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \dots + \frac{1}{R_n}.$$

Hvis tre modstande med modstande $10.5 \, \Omega$, $12.5 \, \Omega$ og $50 \, \Omega$ er forbundet parallelt, er den samlede modstand

$$\frac{1}{R} = \frac{1}{10.5 \, \Omega} + \frac{1}{12.5 \, \Omega} + \frac{1}{50 \, \Omega} = 0.1952 \, \Omega^{-1}$$

$$R = \frac{1}{0.1952 \, \Omega^{-1}} = 5.1219 \, \Omega.$$

Skriv en funktion, der, givet en liste af modstande, returnerer den samlede modstand. Du kan antage, at der er mindst én modstand, og at alle modstande er positive.

Funktionens opførsel er vist nedenfor.

```
>>> parallel_resistors([10.5, 12.5, 50.0])
5.121951219512195
```

Filnavnet og kravene er:

parallel_resistors.py

`parallel_resistors(resistances)`

Parallel resistors.

Parameters:

- `resistances` `list[float]` A non-empty list of positive resistances.

Returns:

- `float` Resistance.

Opgave 4: Graf-transponering (Graph Transpose)

En graf har knuder, der hver er identificeret med et heltal. Knuder er forbundet med kanter, der går fra en knude til en anden. Transponeringen af en graf foretages ved at vende retningen på alle kanter. Betragt for eksempel grafen til venstre. Dens transponering vises til højre.



Vi repræsenterer grafen ved hjælp af en *dictionary*. Hver *key* er en knude med udgående kanter. For hver *key* er den tilknyttede *value* en liste over knuder, som kanterne går til. Knuderne i listerne er i stigende rækkefølge. Hvis en knude ikke har nogen udgående kanter, optræder den ikke som en *key*. For eksempel er grafen til venstre repræsenteret som

```
graph = {
    2: [1],
    1: [2, 4],
    3: [1, 2],
}
```

Skriv en funktion, der, givet en graf, returnerer dens transponerede form i repræsentationen beskrevet ovenfor. Bemærk, at du ikke kan antage, at knuderne er nummereret fortløbende — en graf kan have knuder som 10, 20, 60 med huller i nummereringen.

Funktionens opførsel er vist nedenfor.

```
>>> graph_transpose(graph)
{1: [2, 3], 2: [1, 3], 4: [1]}
```

Filnavnet og kravene er:

graph_transpose.py

`graph_transpose(graph)`

Compute the transpose of a graph.

Parameters:

- `graph` `dict` A directed graph.

Returns:

- `dict` The transposed graph.

Opgave 5: Ulige gennemsnit (Odd Means)

Givet et 2D-array med tal, vil vi beregne gennemsnittet af hver ulige kolonne. Ulige kolonner er den første, tredje, femte og så videre. Betragt nedenstående array.

$$\begin{bmatrix} 5.5 & 2.2 & 3.3 & 2.2 & 3.3 \\ 5.8 & 7 & 8 & 2.2 & 3.3 \\ 10 & 0.9 & 10 & 1.1 & 1.2 \end{bmatrix}$$

Arrayet har fem kolonner, og de ulige kolonner er den første, tredje og femte kolonne. Gennemsnittet for den første kolonne er $(5.5 + 5.8 + 10)/3 = 7.1$. Gennemsnittet for den tredje kolonne er $(3.3 + 8 + 10)/3 = 7.1$. Gennemsnittet for den femte kolonne er $(3.3 + 3.3 + 1.2)/3 = 2.6$. De ulige gennemsnit er altså $[7.1 \quad 7.1 \quad 2.6]$.

Skriv en funktion, der, givet et 2D NumPy-array, returnerer et NumPy-array, der indeholder gennemsnittene for de ulige kolonner.

Funktionens opførsel er vist nedenfor.

```
>>> import numpy as np
>>> M = np.array([[5.5, 2.2, 3.3, 2.2, 3.3],
...               [5.8, 7, 8, 2.2, 3.3],
...               [10, 0.9, 10, 1.1, 1.2]])
>>> odd_means(M)
array([7.1, 7.1, 2.6])
```

Filnavnet og kravene er:

odd_means.py

`odd_means(M)`

Compute the means of odd columns.

Parameters:

- `M` `numpy.ndarray` 2D array.

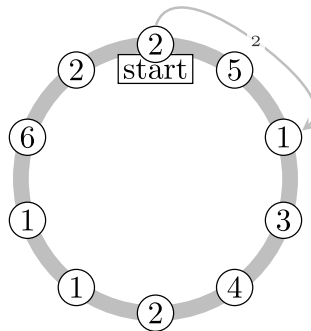
Returns:

- `numpy.ndarray` The means of odd columns.

Opgave 6: Navigationssporing (Navigation Tracking)

Nogle positioner til navigation er arrangeret i en cirkel, hvor en position er udpeget som startposition. Du bevæger dig fremad det antal felter, der er angivet på din aktuelle position, med uret. Det er ikke tilladt at genbesøge en position, så du stopper, før det sker. Du vil gerne kende værdierne for de besøgte positioner i den rækkefølge, de blev besøgt.

Se for eksempel illustrationen nedenfor.



Startpositionen siger, at du skal bevæge dig to trin fremad. Derefter lander du på positionen med nummer 1, som angivet med pilen. Efter at have bevæget dig et skridt fremad, lander du på 3. Når du bevæger dig tre skridt fremad, lander du på 1. Når du bevæger dig et skridt fremad, lander du igen på 1, og når du bevæger dig endnu et skridt fremad, lander du på 6. Efter at have bevæget dig seks skridt fremad, lander du på 4. Derfra bevæger du dig fire skridt og lander på 6, men denne position er allerede blevet besøgt, og positionen med nummer 4 var den sidste position, der blev besøgt. Så positionerne i den rækkefølge, de blev besøgt, er: 2, 1, 3, 1, 1, 6, 4.

Skriv en funktion, der som input tager en liste med heltal, der repræsenterer positionerne, hvor det første element er startpositionen. At øge indekset svarer til at bevæge sig med uret i cirklen. Funktionen skal returnere en liste over de besøgte positioner i den rækkefølge, de blev besøgt. Du kan antage, at der er mindst to positioner, og at alle positioner indeholder positive heltal.

Funktionens opførsel for eksemplet ovenfor er vist nedenfor.

```
>>> navigation_tracking([2, 5, 1, 3, 4, 2, 1, 1, 6, 2])
[2, 1, 3, 1, 1, 6, 4]
```

Filnavnet og kravene er:

navigation_tracking.py

```
navigation_tracking(instructions)
```

Track navigation until repetition.

Parameters:

- `instructions` `list[int]` List of positive integers representing positions.

Returns:

- `list` The positions visited in the order they were visited.

Opgave 7: Sætningskompleksitet (Sentence Complexities)

Du får en tekstfil med sætninger på separate linjer. Hver sætning består af ord, der er sammensat af bogstaver fra det engelske alfabet, adskilt af enkelte mellemrum, og tegnsætning (.,!?) direkte efter nogle ord. Vi definerer en sætnings kompleksitet som:

$$m = (\text{antal ord}) + (\text{antal bogstaver i det længste ord})$$

og vi ønsker at beregne kompleksiteten for alle sætninger i filen.

Betragt for eksempel følgende fil.

files/sentences_1.txt

```
The cat sleeps.  
This sentence is quite long and descriptive.  
Hello world!  
Programming is fun, sometimes challenging.
```

Den første sætning har tre ord, og det længste ord har seks bogstaver (*sleeps*), så dens kompleksitet er $3 + 6 = 9$. Den anden sætning har syv ord, det længste ord har elleve bogstaver (*descriptive*), så dens kompleksitet er $7 + 11 = 18$. Den tredje sætning har to ord, hver med fem bogstaver, så dens kompleksitet er $2 + 5 = 7$. Den fjerde sætning har en kompleksitet på $5 + 11 = 16$. Kompleksiteten for alle sætninger i filen er derfor 9, 18, 7 og 16.

Skriv en funktion, der tager et filnavn som input og returnerer en liste over kompleksiteten af hver sætning i filen. Hvis der er tomme linjer i filen, skal de ignoreres. En tom fil skal resultere i en tom liste.

Funktionen skal fungere som vist nedenfor.

```
>>> sentence_complexities('files/sentences_1.txt')  
[9, 18, 7, 16]
```

Filnavnet og kravene er:

sentence_complexities.py

```
sentence_complexities(filename)
```

Complexity values for all sentences in a file.

Parameters:

- `filename` `str` Path to a file with one sentence per line.

Returns:

- `list` Complexity values for all sentences in the file.

Opgave 8: Cykellygte (Bike Light)

En cykellygte har en *off* tilstand og tre tændte tilstande: *weak*, *strong*, og *flash*. Den styres af en enkelt knap, der reagerer forskelligt på lange og korte tryk.

Lygten starter i *off* tilstand. Et langt tryk tænder lygten i *weak* tilstand. Når lygten er tændt, skifter et kort tryk mellem de tændte tilstande i rækkefølgen: *weak*, *strong*, og *flash*. For at slukke lygten fra en hvilken som helst tændt tilstand bruges et langt tryk. Når lygten tændes igen, starter det i *weak* tilstand, uanset hvilken tilstand lygten var i, da det blev slukket. Når lygten er slukket, ændrer et kort tryk ikke dets tilstand.

Skriv en klasse `BikeLight` der repræsenterer en cykellygte. Konstruktøren skal ikke have nogen argumenter og initialisere lygten i slukket tilstand. Klassen skal have en metode `long_press` der tænder eller slukker lygten. Metoden skal returnere tilstanden som en streng. Klassen skal også have en metode `short_press` der skifter mellem tilstandene, når lygten er tændt. Metoden skal returnere tilstanden som en streng.

Klassens opførsel vises nedenfor.

```
>>> light = BikeLight()
>>> light.long_press()
'weak'
>>> light.short_press()
'strong'
>>> light.short_press()
'flash'
>>> light.long_press()
'off'
>>> light.long_press()
'weak'
```

Filnavnet og kravene er:

bike_light.py

`BikeLight()`

A bike light with multiple modes.

`__init__()`

Initialize the bike light in off mode.

`long_press()`

Turn the light on or off.

Returns:

- `str` The new mode.

`short_press()`

Cycle through modes when the light is on.

Returns:

- `str` The new mode.

Opgave 9: Kuffert (Suitcase)

Skriv en klasse `Suitcase` der repræsenterer en kuffert, hvor konstruktøren tager en vægtgrænse som argument. Klassen skal have en metode, `pack_item` der tager en ting og dets vægt som argument. Den pakker tingene i kufferten, hvis den samlede vægt af de ting, der allerede er i kufferten, og den nye vægt ikke overskrider grænsen. Metoden `currently_packed` returnerer en liste over ting i kufferten i den rækkefølge, de blev pakket. Metoden `__add__` kombinerer to `Suitcase`'s til en ny `Suitcase` med en vægtgrænse svarende til summen af de to, pakket med alle ting fra den første kuffert efterfulgt af ting fra den anden kuffert, hver i deres oprindelige pakkerækkefølge. Se eksemplet nedenfor.

```
>>> carry_on = Suitcase(7.0)
>>> check_in = Suitcase(23.0)
>>> carry_on.pack_item('Laptop', 1.5)
True
>>> check_in.pack_item('Clothes', 5.0)
True
>>> check_in.pack_item('E-Bike', 20.0)
False
>>> all_luggage = carry_on + check_in
>>> all_luggage.currently_packed()
['Laptop', 'Clothes']
```

Der oprettes to kufferter med en vægtgrænse på 7 kg for *carry-on* og 23 kg for *check-in* kuffert. En *laptop* på 1.5 kg tilføjes til *carry-on* kufferten, og 5 kg *clothes* tilføjes til *check-in* kufferten. Det er ikke muligt at tilføje en 20 kg *E-bike* til *check-in* kufferten, da $5 + 20 = 25$ er større end 23. Til sidst lægges de to kufferter sammen, og indholdet af den nye kombinerede kuffert vises.

Filnavnet og kravene er:

suitcase.py

`Suitcase()`

A suitcase that can hold items up to a weight limit.

`__init__(weight_limit)`

Initialize the suitcase with a maximum weight.

Parameters:

- `weight_limit` `float` The weight limit of the suitcase.

`pack_item(item, weight)`

Pack an item if the total weight of items in the suitcase after packing it does not exceed the weight limit.

Parameters:

- `item` `str` The name of the item.
- `weight` `float` The weight of the item.

Returns:

- `bool` `True` if the item was packed, `False` otherwise.

`__add__(other)`

Combine two suitcases into a new one.

Parameters:

- `other` `Suitcase` Another suitcase.

Returns:

- `Suitcase` A new suitcase with a combined limit and items.

`currently_packed()`

Return the items currently packed in the suitcase.

Returns:

- `list` List of items.

Opgave 10: Butiksstatus (Shop Status)

En butik har åbent mandag til fredag fra kl. 09:00 til 17:00. Lørdag og søndag er butikken lukket. Vi vil gerne vide, om butikken har åbent, og hvornår den åbner eller lukker næste gang.

Skriv en funktion, der tager to argumenter: en ugedag og et timetal (et heltal fra 0 til 23). Funktionen skal returnere, om butikken er åben på det tidspunkt, hvilken dag dens status ændres næste gang, og hvilket timetal ændringen finder sted. På hverdage er butikken allerede åben, hvis klokken er præcis 9, og den er allerede lukket, hvis klokken er præcis 17.

Du kan bruge følgende linje kode i din løsning:

```
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
```

Se eksemplet nedenfor.

```
>>> shop_status('Monday', 9)
(True, 'Monday', 17)
```

Her vil vi gerne vide butikkens status mandag kl. 9. Butikken er åben på dette tidspunkt. Status ændres mandag kl. 17, når butikken lukker.

Filnavnet og kravene er:

shop_status.py

`shop_status(day, hour)`

Determine if the shop is open or closed.

Parameters:

- `day` `str` The day of the week.
- `hour` `int` The hour of the day (0-23).

Returns:

- `tuple[bool, str, int]`
 - A tuple containing:
 - Whether the shop is currently open.
 - The day of the week when the status will change next.
 - The hour when the status will change next.