

Eksamen sommeren 2026

Kursusnavn:	Programmering
Kursusnummer:	02002 og 02003
Eksamenstidspunkt:	30. maj 2026
Hjælpemidler tilladt:	Alle hjælpemidler, intet internet
Eksamenens varighed:	4 timer
Vægtning:	Alle opgaver har samme vægt
Antal opgaver:	10
Antal sider:	12

Contents

- Eksamensinstruktioner
- Opgave 1: Pendulperiode (Pendulum Period)
- Opgave 2: Antal negative tal (Count Negatives)
- Opgave 3: Cifreantal (Digit Count)
- Opgave 4: Talrækkeforskel (Sequence Difference)
- Opgave 5: Afstand mellem timepunkter (Hour Point Distance)
- Opgave 6: Købsmuligheder (Buying Options)
- Opgave 7: Adgangskodeanalyse (Password Analysis)
- Opgave 8: Intensitetsminutter (Intensity Minutes)
- Opgave 9: Kortspiller (Card Player)
- Opgave 10: Beholder (Container)

Eksamensinstruktioner

Eksamensmateriale

Eksamensmaterialet er en enkelt zip-fil, som du skal udpakke til en mappe på din computer. Zip-filen indeholder eksamensoppgaven som et pdf-dokument på engelsk `2026_05_exam_English.pdf` og det samme dokument på dansk `2026_05_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 `2026_05_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 `2026_05_exam`. Hvis du bruger VS Code, skal du derfor klikke på `File` → `Open Folder...` og vælge mappen `2026_05_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.

- `buying_options.py`
- `card_player.py`
- `container.py`
- `count_negatives.py`
- `digit_count.py`
- `hour_point_distance.py`
- `intensity_minutes.py`
- `password_analysis.py`
- `pendulum_period.py`
- `sequence_difference.py`

Opgave 1: Pendulperiode (Pendulum Period)

Perioden T (i sekunder) for et pendul er

$$T = 2\pi\sqrt{\frac{L}{g}}$$

hvor L er pendulets længde (i meter) og $g = 9.81\text{m/s}^2$ er tyngdeaccelerationen.

For eksempel har et pendul med en længde på 1.5 meter en periode på $2\pi\sqrt{\frac{1.5}{9.81}} = 2.4569$ sekunder.

Skriv en funktion, der tager pendulets længde i meter som input og returnerer dets periode i sekunder.

Funktionens opførsel vises nedenfor.

```
>>> pendulum_period(1.5)
2.456919878869914
```

Filnavnet og kravene er:

pendulum_period.py

`pendulum_period(length)`

Return the time period T of a simple pendulum of given length L .

Parameters:

- `length` `float` length of the pendulum in meters.

Returns:

- `float` Time period in seconds.

Opgave 2: Antal negative tal (Count Negatives)

En liste indeholder heltal. Hvert heltal kan være positivt, negativt eller nul. Vi vil tælle, hvor mange heltal i listen der er negative. Hvis listen er tom, er antallet 0.

Betragt for eksempel listen

```
[3, 8, -2, 0, -10, 17, -1, -1, -2]
```

Den indeholder i alt fem negative heltal, nemlig -2, -10, -1, -1 og -2. Derfor er antallet 5.

Skriv en funktion, der tager en liste som input og returnerer antallet.

Funktionens opførsel vises nedenfor.

```
>>> count_negatives([3, 8, -2, 0, -10, 17, -1, -1, -2])
5
```

Filnavnet og kravene er:

count_negatives.py

```
count_negatives(values)
```

Return the number of negative integers in the list.

Parameters:

- `values` `list[int]` A list of integers.

Returns:

- `int` The count of negative integers in the list.

Opgave 3: Cifreantal (Digit Count)

Givet et positivt heltal og et ciffer fra 0 til 9, vil vi tælle, hvor mange gange cifferet forekommer, når man skriver alle tal fra 1 op til det givne heltal.

Betragt for eksempel tallet 13 og cifferet 1. I talrækken

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

forekommer cifferet 1 i fem tal: 1, 10, 11, 12 og 13. Det forekommer to gange i 11. I alt forekommer cifferet 1 altså 6 gange.

Skriv en funktion, der tager et positivt heltal og et ciffer som input. Funktionen skal returnere, hvor mange gange cifferet forekommer i rækkefølgen fra 1 til det givne tal.

Funktionens opførsel vises nedenfor.

```
>>> digit_count(13, 1)
6
```

Filnavnet og kravene er:

digit_count.py

`digit_count(N, d)`

Counts occurrence of a digit in a sequence.

Parameters:

- `N` `int` A positive number defining a sequence from 1 to N.
- `d` `int` A digit 0-9.

Returns:

- `int` The count of occurrences of d in the sequence from 1 to N.

Opgave 4: Talrækkeforskel (Sequence Difference)

Givet to talsekvenser af samme længde ønsker vi at beregne forskellen mellem sekvenserne. Forskellen d mellem sekvenserne a_i og b_i defineres som summen af de absolutte forskelle mellem de tilsvarende elementer

$$d = \sum_{i=1}^n |a_i - b_i|.$$

Betragt for eksempel en talrække 1.2, 2.4, 3.6 og en talrække 4.2, 0.2, 6.6. Forskellen er

$$d = |1.2 - 4.2| + |2.4 - 0.2| + |3.6 - 6.6| = 3.0 + 2.2 + 3.0 = 8.2.$$

Skriv en funktion, der tager to lister med tal som input, hvor hver liste repræsenterer en talrække. Funktionen skal returnere forskellen. Du kan antage, at de to lister har samme længde.

Funktionens opførsel vises nedenfor.

```
>>> a = [1.2, 2.4, 3.6]
>>> b = [4.2, 0.2, 6.6]
>>> sequence_difference(a, b)
8.2
```

Filnavnet og kravene er:

sequence_difference.py

`sequence_difference(a, b)`

Calculates the difference between two sequences.

Parameters:

- `a` `list[float]` The first sequence of numbers.
- `b` `list[float]` The second sequence of numbers.

Returns:

- `float` The difference between the two sequences.

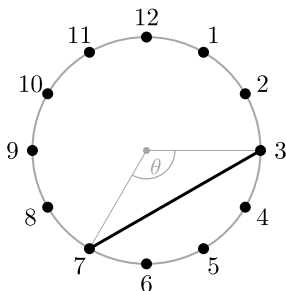
Opgave 5: Afstand mellem timepunkter (Hour Point Distance)

Et urskive har punkter nummereret fra 1 til 12 på en cirkel med radius 1. Vi vil finde afstanden mellem to vilkårlige af disse timepunkter.

En tilgang er denne:

- Find det korteste antal trin mellem de to timepunkter (med uret eller mod uret).
- Konverter trinene til en vinkel θ , hvor hvert trin er $\frac{2\pi}{12}$ radianer (en tolvtedel af den fulde cirkel).
- Brug formlen for afstanden mellem to punkter på enhedscirklen $d = 2 \sin\left(\frac{\theta}{2}\right)$.

Betragt for eksempel timepunkterne 7 og 3, vist nedenfor.



Med udgangspunkt i 7 er den korteste vej til 3 at bevæge sig 4 trin mod uret. Så vinklen mellem de to timepunkter er $\theta = 4 \cdot \frac{2\pi}{12}$, og afstanden er

$$d = 2 \sin\left(\frac{4 \cdot \frac{2\pi}{12}}{2}\right) = 1.732.$$

Skriv en funktion, der tager to timepunkter (heltal fra 1 til 12) som indgang og returnerer afstanden mellem dem.

Funktionens opførsel vises nedenfor.

```
>>> hour_point_distance(7, 3)
1.7320508075688772
```

Filnavnet og kravene er:

hour_point_distance.py

```
hour_point_distance(h1, h2)
```

Calculates the distance between two hour-points on a clock face.

Parameters:

- `h1` `int` The first hour-point (1-12).
- `h2` `int` The second hour-point (1-12).

Returns:

- `float` The distance between the two hour-points.

Opgave 6: Købsmuligheder (Buying Options)

En *dictionary* indeholder varer og deres pris pr. enhed. Givet et beløb ønsker vi en *dictionary* med varer og antal hele enheder, der kan købes for det beløb.

Skriv en funktion, der tager en *dictionary* med priser og et beløb som input. Funktionen skal returnere en *dictionary* med varer og antal enheder, der kan købes. Udelad varer, hvor beløbet ikke er stort nok til at købe en hel enhed. Du må ikke ændre den oprindelige *dictionary*.

Betragt for eksempel denne *dictionary*

```
prices = {'mango': 4.5, 'banana': 3.5, 'chocolate': 10.0}
```

og beløbet 15.0.

Med dette beløb kan du købe 3 mangoer ($15.0/4.5 = 3.33$, hvilket er nok til 3, men ikke 4), eller 4 bananer (da $15.0/3.5 = 4.28$ rundes ned til 4), eller 1 chokolade ($15.0/10.0 = 1.5$, hvilket rundes ned til 1). Den returnerede *dictionary* kan ses nedenfor.

```
>>> buying_options(prices, 15.0)
{'mango': 3, 'banana': 4, 'chocolate': 1}
```

Filnavnet og kravene er:

buying_options.py

```
buying_options(prices, amount)
```

Determine the number of units that can be bought for each item given an amount.

Parameters:

- `prices` `dict` A dictionary with items as keys and their prices as values.
- `amount` `float` The total amount available to spend.

Returns:

- `dict` A dictionary with items as keys and the number of units that can be bought as values.

Opgave 7: Adgangskodeanalyse (Password Analysis)

En adgangskode består af cifre og bogstaver fra det engelske alfabet. Vi vil tælle antallet af bogstaver og cifre i en adgangskodestreng.

For eksempel har adgangskoden "PASS123word" 8 bogstaver (P, A, S, S, w, o, r, d) og 3 cifre (1, 2, 3).

Skriv en funktion, der tager en adgangskodestreng som input og returnerer en tuple, der indeholder antallet af bogstaver og antallet af cifre i adgangskoden. Du kan antage, at adgangskoden kun indeholder engelske bogstaver (A–Z, a–z) og cifre (0–9).

```
>>> password_analysis("PASS123word")
(8, 3)
```

Filnavnet og kravene er:

password_analysis.py

`password_analysis(password)`

Analyze a password to count letters and digits.

Parameters:

- `password` `str` The password string to analyze.

Returns:

- `tuple[int, int]` The number of letters and the number of digits.

Opgave 8: Intensitetsminutter (Intensity Minutes)

En fil indeholder registreringer af daglige aktiviteter. Hver linje i filen indeholder to heltal: antallet af minutter for en aktivitet og dens intensitetsniveau. Intensitetsniveauet er 1 for lav intensitet, 2 for moderat intensitet og 3 for høj intensitet. Det samlede antal intensitetsminutter for en dag er summen af minutter fra aktiviteter på niveau 2 og niveau 3, hvor minutter fra aktiviteter på niveau 3 tæller dobbelt.

Skriv en funktion, der tager et filnavn som input og returnerer det samlede antal intensitetsminutter for dagen.

Betragt for eksempel følgende fil:

files/activities_1.txt

```
46 2
8 3
21 2
15 1
3 2
```

Der er fem aktiviteter i filen. Tre aktiviteter er på niveau 2 med varigheder på 46, 21 og 3 minutter, hvilket bidrager med $46 + 21 + 3 = 70$ intensitetsminutter. Én aktivitet er på niveau 3 med en varighed på 8 minutter, hvilket bidrager med $2 \cdot 8 = 16$ intensitetsminutter. Aktiviteten på niveau 1, med en varighed på 15 minutter, bidrager ikke til intensitetsminutterne. Derfor er det samlede antal intensitetsminutter $70 + 16 = 86$, som vist nedenfor.

```
>>> intensity_minutes('files/activities_1.txt')
86
```

Filnavnet og kravene er:

intensity_minutes.py

```
intensity_minutes(filename)
```

Calculates total intensity minutes from activity records in a file.

Parameters:

- `filename` `str` The name of the file containing activity records.

Returns:

- `int` The total intensity minutes for the day.

Opgave 9: Kortspiller (Card Player)

En kortspiller starter med 0 point og en håndfuld kort. Hvert kort har en unik værdi mellem 1 og 100. Når en modstander spiller et kort, tjekker spilleren, om vedkommendes hånd indeholder et eller flere kort med en højere værdi. Hvis det er tilfældet, bruger spilleren det mindste af disse kort og får point svarende til summen af det kort og modstanderens kort. Hvis ikke, bruger spilleren det mindste kort i sin hånd og får ingen point.

Skriv en klasse `CardPlayer` med en konstruktør, der tager en liste over heltal, der repræsenterer spillerens hånd. Klassen skal have en metode `play` der tager et heltal, der repræsenterer kortet spillet af modstanderen. Metoden skal vælge det passende kort at bruge, fjerne det fra hånden og returnere spillerens samlede point efter at have brugt kortet. Du kan antage, at alle kort i spillet (inklusive modstanderens kort) har forskellige værdier, og at `play` kun kaldes, når spilleren har mindst ét kort på hånden.

Opførslen er illustreret nedenfor for en starthånd med kortene 42, 30 og 50. Først spiller modstanderen 35, og spilleren bruger 42 og får $35 + 42 = 77$ point. Dernæst spiller modstanderen 61, og spilleren bruger 30, hvilket ikke giver yderligere point. Til sidst spiller modstanderen 10, og spilleren bruger 50, hvilket giver $10 + 50 = 60$ point. Det samlede antal point er derfor $77 + 60 = 137$, som vist nedenfor.

```
>>> player = CardPlayer([42, 30, 50])
>>> player.play(35)
77
>>> player.play(61)
77
>>> player.play(10)
137
```

Filnavnet og kravene er:

card_player.py

`CardPlayer()`

A card player playing cards against opponent's cards.

`__init__(hand)`

Initialize an card player with a hand of cards.

Parameters:

- `hand` `list[int]` A list of integers representing the player's hand.

`play(opponent_card)`

Play a card against an opponent's card.

Parameters:

- `opponent_card` `int` The value of the opponent's played card.

Returns:

- `int` The current points after playing the card.

Opgave 10: Beholder (Container)

En beholder har en kapacitet, som er et heltal, og den inderholder en mængde mellem 0 og kapaciteten.

Skriv en klasse `Container` med en konstruktør, der tager kapaciteten som input og opretter en tom beholder. Metoden `fill` tager en heltalsværdi, der skal fyldes i beholderen. Metoden fylder beholderen, men overskrider aldrig dens kapacitet. Den returnerer den mængde, der faktisk er fyldt. Metoden `remove` tager en heltalsværdi, der skal fjernes fra beholderen. Metoden fjerner fra beholderen, men aldrig under 0. Den returnerer den mængde, der faktisk er fjernet. Metoden `__str__` returnerer en strengrepræsentation af beholderen i formatet `"Container: x/y"`, hvor `x` er den aktuelle mængde i beholderen og `y` er kapaciteten.

Opførslen er illustreret nedenfor med en beholder med en kapacitet på 10. Først fyldes 7. Derefter forsøges det at fylde 5, men der er kun plads til 3. Til sidst forsøges det at fjerne 14, men der er kun 10 til rådighed.

```
>>> container = Container(10)
>>> str(container)
'Container: 0/10'
>>> container.fill(7)
7
>>> str(container)
'Container: 7/10'
>>> container.fill(5)
3
>>> str(container)
'Container: 10/10'
>>> container.remove(14)
10
>>> str(container)
'Container: 0/10'
```

Filnavnet og kravene er:

container.py

`Container()`

A container with a capacity that can be filled and emptied.

`__init__(capacity)`

Initialize the container with a given capacity.

Parameters:

- `capacity` `int` The capacity of the container.

`fill(amount)`

Fill the container with a given amount.

Parameters:

- `amount` `int` The amount to fill.

Returns:

- `int` The actual amount filled.

`remove(amount)`

Remove a given amount from the container.

Parameters:

- `amount` `int` The amount to remove.

Returns:

- `int` The actual amount removed.

`__str__()`

A string representation of the container.

Returns:

- `str` A string in the format "Container: x/y".