

Eksamen sommer 2025

| | |
|-----------------------|-----------------------------------|
| Kursets navn: | Programmering |
| Kursusnummer: | 02002 og 02003 |
| Eksamensdato: | 28. maj 2025 |
| Hjælpemidler tilladt: | Alle hjælpemidler, intet internet |
| Eksamens varighed: | 4 timer |
| Vægtning: | Alle opgaver har samme vægt |
| Antal opgaver: | 10 |
| Antal sider: | 13 |

Contents

- Eksamensinstruktioner
- Opgave 1: Polygonens areal (Polygon Area)
- Opgave 2: Modificeret blackjack (Modified Blackjack)
- Opgave 3: Løbehastighed (Running Speed)
- Opgave 4: Fotodetaljer (Photo Details)
- Opgave 5: Eksperimentnumre (Experiment Numbers)
- Opgave 6: Hoppende bold (Bouncing Ball)
- Opgave 7: Talstatistik (Number Statistics)
- Opgave 8: To gennemsnit (Two Means)
- Opgave 9: Bingokort (Bingo Card)
- Opgave 10: Informativt bingokort (Informative Bingo Card)

Eksamensinstruktioner

Forudsætninger

For at kunne løse eksamensopgaverne skal du have en computer med Python installeret. Alle eksamensopgaver kan løses i enten IDLE eller VS Code.

Eksamensmateriale

Eksamensmaterialet består af en enkelt zip-fil. Du skal pakke denne fil ud i en mappe på din computer. Zip-filen indeholder eksamensteksten som et PDF-dokument på engelsk `2025_05_exam_English.pdf` og det samme dokument på dansk `2025_05_exam_Danish.pdf` (dette dokument). Zip-filen indeholder også en mappe `2025_05_exam` med følgende indhold:

- En tom Python-fil for hver opgave, `<opgave_navn>.py`, hvor `<opgave_navn>` er navnet på opgaven. Det er i disse filer, du skal skrive dine løsninger og aflevere dem ved eksamens afslutning.
- En Python-fil for hver opgave, `test_task_<n>_<opgave_navn>.py`, hvor `<n>` er opgavens nummer, og `<opgave_navn>` er opgavens navn. Disse indeholder kode, der tester, om din løsning har den korrekte opførsel i forhold til eksemplet i eksamensteksten. For at være sikker på, at du bruger testene efter hensigten, skal du ikke redigere disse filer.
- En Python-fil `test_tasks_all.py`, der kører alle testfiler.
- En mappe `files`, der indeholder datafiler, som er nødvendige for at teste opgaver, der involverer filer, hvis der er nogen.

Løsning af eksamensopgaver

Hvis du bruger VS Code, skal du starte med at gå til `File` → `Open Folder...` og vælge mappen `2025_05_exam` i den mappe, du pakkede ud som beskrevet ovenfor.

Når du løser eksamensopgaverne, skal du følge instruktionerne i eksamensteksten. Du kan teste dine løsninger ved at køre de medfølgende test-scripts. For at test-scriptene skal virke, skal dine løsninger ligge i den samme mappe som test-scriptene.

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

Dine løsninger bør kun bruge de værktøjer, der er blevet undervist i på kurset. Løsninger, der importerer andre moduler end `math`, `numpy`, `os` eller `matplotlib`, vil ikke blive bedømt. De medfølgende test-scripts tjekker ikke for dette, så det er dit ansvar at sikre, at dine løsninger kun bruger de tilladte moduler.

Evaluerings af eksamen

Vi kører en række ekstra tests på hver af dine løsninger for at tjekke, om den opfører sig som specificeret i opgaven. Andelen af korrekte tests er scoren for hver opgave. Gennemsnittet af disse test-scoringer giver den samlede score.

En løsning er forket, hvis den medfølgende test fejler. Det kan skyldes, at filen eller funktionen er navngivet forkert. Men hvis en medfølgende test består, garanterer det ikke, at løsningen er korrekt for vores yderligere tests.

Aflevering af løsninger

For at aflevere dine løsninger skal du uploade dine Python-filer med løsninger til Digital Exam-systemet. I Digital Exam-systemet kan filer indsendes som enten *hoveddokument* eller *bilag*. Du kan uploade en hvilken som helst af dine løsninger som hoveddokument og resten som bilag.

Du skal aflevere præcis følgende filer:

- `bingo_card.py`
- `bouncing_ball.py`
- `experiment_numbers.py`
- `modified_blackjack.py`
- `number_statistics.py`
- `photo_details.py`
- `polygon_area.py`
- `running_speed.py`
- `two_means.py`

Enhver afleveret fil, der ikke er på listen ovenfor, vil ikke blive taget i betragtning i din vurdering.

Opgave 1: Polygonens areal (Polygon Area)

Arealet af en regulær polygon med n sider, der hver har længden s , er givet ved

$$A = \frac{ns^2}{4 \tan\left(\frac{\pi}{n}\right)}.$$

For eksempel er arealet af en regulær polygon med fem sider (en femkant) og sidelængden 4.31

$$A = \frac{5 \cdot 4.31^2}{4 \tan\left(\frac{\pi}{5}\right)} = \frac{92.8805}{2.9062} = 31.9598.$$

Skriv en funktion, der tager antallet af sider og sidelængden som input og returnerer arealet af den regulære polygon.

Den ønskede opførsel for en regulær polygon med fem sider og en sidelængde på 4.13 er vist nedenfor.

```
>>> polygon_area(5, 4.31)
31.9597602410807
```

Filnavnet og kravene er:

polygon_area.py

`polygon_area(n, s)`

Calculates the area of a regular polygon.

Parameters:

- `n` `int` The number of sides.
- `s` `float` The length of each side.

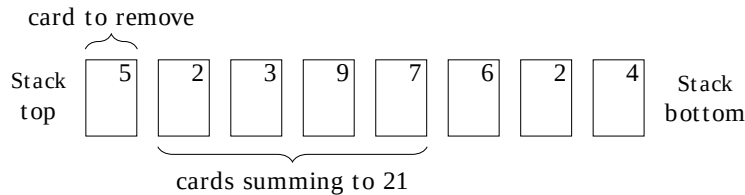
Returns:

- `float` The area of the polygon.

Opgave 2: Modificeret blackjack (Modified Blackjack)

Du har en stak blandede spillekort, som hver har en værdi mellem 1 og 11. Du vil gerne finde ud af, om det er muligt at fjerne n kort fra toppen af stakken, så de k kort, der nu ligger på toppen af stakken, har værdier, hvis sum er præcis 21.

Betragt for eksempel sekvensen 5, 2, 3, 9, 7, 6, 2, 4, hvor den første værdi er det øverste kort, som vist nedenfor. Hvis vi ikke fjerner nogen kort, har de fire øverste kort en sum på $5 + 2 + 3 + 9 = 19$, hvilket er under 21, og hvis vi tilføjer det femte kort, kommer summen op på $5 + 2 + 3 + 9 + 7 = 26$, hvilket er over 21, så $n = 0$ er ikke en løsning. Hvis vi fjerner et kort, giver de fire øverste kort summen $2 + 3 + 9 + 7 = 21$. Så kombinationen $n = 1$ og $k = 4$ er en løsning.



Skriv en funktion, der som input tager en liste med heltal, der repræsenterer værdierne af kortene i stakken. Funktionen skal returnere to tal, n og k . Her er n antallet af kort, der skal fjernes, og k er antallet af kort, hvis sum er 21. Hvis der ikke findes nogen løsning, returneres -1 og -1 . Hvis der findes flere løsninger, returneres den med det mindste n .

Den ønskede opførsel for eksemplet er vist nedenfor.

```
>>> modified_blackjack([5, 2, 3, 9, 7, 6, 2, 4])
(1, 4)
```

Filnavnet og kravene er:

modified_blackjack.py

`modified_blackjack(cards)`

Finds the solution given a sequence of values.

Parameters:

- `cards` `list` A sequence of integers representing the values of cards in a stack.

Returns:

- `tuple` n and k according to the rules.

Opgave 3: Løbehastighed (Running Speed)

Løbepace angives i minutter pr. kilometer. For eksempel betyder en pace på 5.5 minutter pr. kilometer, at man løber en kilometer på 5.5 minutter. Det inverse af pace er hastighed.

Vi ønsker at konvertere en given pace til hastigheden i kilometer i timen og hastigheden i meter pr. sekund.

For eksempel er hastigheden for en pace på 5.5 minutter pr. kilometer

$$\frac{1}{5.5} \frac{\text{km}}{\text{min}} = \frac{1}{5.5} \cdot 60 \frac{\text{km}}{\text{h}} = 10.9 \frac{\text{km}}{\text{h}}$$

og

$$\frac{1}{5.5} \frac{\text{km}}{\text{min}} = \frac{1}{5.5} \cdot \frac{1000}{60} \frac{\text{m}}{\text{s}} = 3.03 \frac{\text{m}}{\text{s}}$$

Skriv en funktion, der tager en pace i minutter pr. kilometer og returnerer hastigheden i kilometer i timen og hastigheden i meter pr. sekund. Den ønskede opførsel for pacen 5.5 minutter pr. kilometer er vist nedenfor.

```
>>> running_speed(5.5)
(10.90909090909091, 3.03030303030307)
```

Filnavnet og kravene er:

running_speed.py

`running_speed(pace)`

Given pace computes speed in km/h and m/s.

Parameters:

- `pace` `float` Pace in min/km.

Returns:

- `tuple` Speed in km/h and m/s.

Opgave 4: Fotodetaljer (Photo Details)

Filnavnet på et foto indeholder datoen, hvor det blev taget, et identifikationsnummer og en filendelse. I filnavnet

`20250305_110031.jpg` er den første del, `20250305`, f.eks. en dato i formatet `YYYYMMDD` efterfulgt af et `_`. Den anden del, `110031`, er et identifikationsnummer for billedet, og `.jpg` er filtypen. Vi er nødt til at udtrække disse detaljer fra filnavnet.

Skriv en funktion, der tager et filnavn i formatet `YYYYMMDD_ID.EXT` og returnerer en *dictionary* med følgende *keys* og *values*:

- `'year'`: de fire tegn, der repræsenterer året (som en streng)
- `'month'`: de to tegn, der repræsenterer måneden (som en streng)
- `'day'`: de to tegn, der repræsenterer dagen (som en streng)
- `'number'`: tegnene i identifikationsnummeret (som en streng)
- `'ext'`: filtypen (som en streng, uden punktum)

Du kan se den ønskede opførsel for strengen `'20250305_110031.jpg'` nedenfor.

```
>>> photo_details('20250305_110031.jpg')
{'year': '2025', 'month': '03', 'day': '05', 'number': '110031', 'ext': 'jpg'}
```

Bemærk, at mens datoformatet altid er `YYYYMMDD`, kan filidentifikatoren og filtypen variere i længde. For eksempel er `'20250305_2151.jpeg'` også et gyldigt filnavn. Du kan gå ud fra, at identifikationsnummeret er et heltal.

Filnavnet og kravene er:

photo_details.py

`photo_details(filename)`

Given a filename, extract the details of the photo.

Parameters:

- `filename` `str` The filename of the photo.

Returns:

- `dict` The details of the photo.

Opgave 5: Eksperimentnumre (Experiment Numbers)

Et antal eksperimenter repræsenteres som en liste, for eksempel

```
['A-1', 'A-3', 'B-2', 'A-4', 'B-1', 'B-3', 'N-4'].
```

Hvert element består af et eksperimentnavn (et enkelt bogstav), en bindestreg og et (positivt) gentagelsesnummer. For eksempel betyder `'A-4'`, at eksperiment A blev gentaget for fjerde gang. Listen er ikke ordnet, ikke alle gentagelser er nødvendigvis til stede, og gentagelsesnumre kan være mere end et ciffer.

Skriv en funktion, der tager listen over eksperimenter som input og returnerer en *dictionary* med eksperimenternes navne som *keys*. Hver *key* skal som *value* have det højeste gentagelsesnummer, der er fundet for det pågældende eksperiment, repræsenteret som et heltal.

Du kan se den ønskede adfærd nedenfor.

```
>>> experiment_numbers(['A-1', 'A-3', 'B-2', 'A-4', 'B-1', 'B-3', 'N-4'])
{'A': 4, 'B': 3, 'N': 4}
```

Filnavnet og kravene er:

experiment_numbers.py

`experiment_numbers(experiments)`

Computes statistics about a list of experiments.

Parameters:

- `experiments` `list` List of experiments.

Returns:

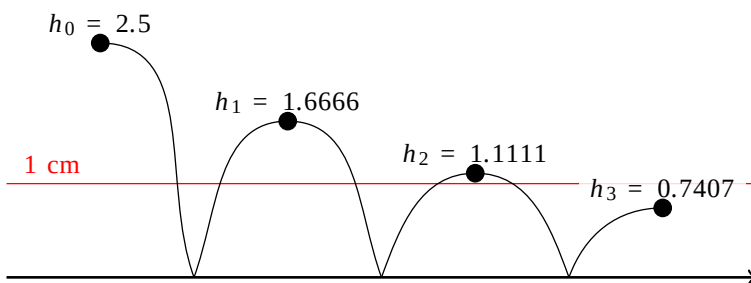
- `dict` Maximum repetitions for each experiment name.

Opgave 6: Hoppende bold (Bouncing Ball)

En bold tabes fra en højde på h_0 centimeter og får lov til at hoppe. Efter hvert hop når den en højde, der er to tredjedele af den forrige højde. Det vil sige

$$h_{n+1} = \frac{2}{3}h_n$$

hvor h_n er højden efter det n^{te} hop. Vi vil gerne finde ud af, hvor mange gange bolden hopper, før den når en højde på under 1 centimeter.



Overvej eksemplet fra illustrationen. Bolden tabes fra en højde på $h_0 = 2.5$ centimeter. Højden efter det første opspring er $h_1 = \frac{2}{3}2.5 = 1.666$ centimeter, efter det andet hop $h_2 = \frac{2}{3}1.666 = 1.1111$ centimeter, og efter det tredje hop $h_3 = \frac{2}{3}1.1111 = 0.7407$ centimeter. Da h_3 er under 1 centimeter, hoppede bolden tre gange, før højden faldt til under 1 centimeter.

Skriv en funktion, der tager den oprindelige højde h_0 som input og returnerer antallet af hop, før højden er under 1 centimeter.

Den ønskede opførsel for en starthøjde på 2.5 centimeter er vist nedenfor.

```
>>> bouncing_ball(2.5)
3
```

Filnavnet og kravene er:

bouncing_ball.py

bouncing_ball(h_0)

Calculates how many bounces a ball makes before bouncing less than 1 centimeter.

Parameters:

- h_0 float The initial height of the ball in centimeters.

Returns:

- int The number of bounces before the ball bounces less than 1 centimeter.

Opgave 7: Talstatistik (Number Statistics)

En bestemt type tekstfil indeholder linjer, og hver linje indeholder heltal adskilt af mellemrum. Givet et tal vil vi gerne vide, hvor mange gange dette tal forekommer i hver linje i filen.

Betragt for eksempel filen `files/numbers_1.txt` med indholdet:

```
7 4 8 12 11 9 18 4 25
9 16 47 2 26 74 57 33
4 7 9 8 3 6 5 8 6 4 3
```

og tallet 8. Tallet 8 forekommer én gang i den første linje, ingen gange i den anden linje og to gange i den sidste linje. Dette resultat kan repræsenteres som listen `[1, 0, 2]`.

Skriv en funktion, der, givet et filnavn og et tal, returnerer listen med antallet af forekomster af dette tal i hver linje i filen. Hvis der er tomme linjer i filen, skal de ignoreres. Du kan gå ud fra, at der er præcis et mellemrum mellem tallene og ingen mellemrum før det første eller efter det sidste tal. Bemærk, at cifret 8 i 18 ikke skal tælles med, da vi tæller hele tal og ikke enkelte cifre.

Det forventede output kan ses i eksemplet.

```
>>> filename = 'files/numbers_1.txt'
>>> number_statistics(filename, 8)
[1, 0, 2]
```

Filnavnet og kravene er:

`number_statistics.py`

`number_statistics(filename, number)`

Compute statistics.

Parameters:

- `filename` `str` The filename.
- `number` `int` The number to check for.

Returns:

- `list` The number of occurrences in each line.

Opgave 8: To gennemsnit (Two Means)

Ud fra nogle tal og en tærskelværdi opretter vi to delmængder: tal, der ligger lige under tærsklen, og tal, der ligger lige over tærsklen. Vi ønsker at finde gennemsnittet af hver delmængde.

Overvej for eksempel tallene

1.5, 8.5, 3.5, 4.5, 5.0, 6.5, 7.5, 2.5, 1.0

og tærsklen 5.5. Tallene lige under tærsklen er 1.5, 3.5, 4.5, 5.0, 2.5 og 1.0, som har et gennemsnit på 3.0. Tallene, der ligger lige over tærsklen, er 8.5, 6.5 og 7.5, som har et gennemsnit på 7.5.

Skriv en funktion, der, givet et array af tal og en tærskel, returnerer gennemsnittet af de tal, der ligger lige under tærsklen, og gennemsnittet af de tal, der ligger lige over tærsklen. Hvis en eller begge delmængder er tomme (dvs. ingen tal under eller over tærsklen), skal du returnere tærskelværdien i stedet for et gennemsnit for den pågældende delmængde.

Den ønskede opførsel er vist nedenfor.

```
>>> import numpy as np
>>> two_means(np.array([1.5, 8.5, 3.5, 4.5, 5.0, 6.5, 7.5, 2.5, 1.0]), 5.5)
(np.float64(3.0), np.float64(7.5))
```

Filnavnet og kravene er:

two_means.py

`two_means(numbers, threshold)`

Computes two means.

Parameters:

- `numbers` `numpy.ndarray` Array of numbers.
- `threshold` `float` Threshold value.

Returns:

- `tuple` Two means.

Opgave 9: Bingokort (Bingo Card)

Skriv en klassedefinition for klassen `BingoCard`, der repræsenterer et kort til et spil bingo. Konstruktøren skal som input tage en liste af numre på kortet. Metoden `match` skal tage et tal som input, og hvis tallet er på kortet, skal den registrere, at tallet er blevet matchet. Den skal returnere `True`, hvis tallet blev matchet, og `False` i modsat fald. Metoden `unmatched` bør returnere `True`, hvis der stadig er numre, der ikke er matchet, og `False`, hvis alle numre er blevet matchet. Metoden `reset` skal rydde de matchede numre, så kortet kan bruges igen.

Du kan antage, at klassen bruges, som bingo typisk spilles. Det vil sige, at `match` ikke vil blive kaldt med det samme nummer mere end én gang, før kortet er nulstillet, et nummer vil ikke optræde to gange på et kort, og kortet vil ikke være tomt. Overvej eksemplet nedenfor.

```
>>> card = BingoCard([15, 27, 73])
>>> card.match(27)
True
>>> card.match(53)
False
>>> card.match(73)
True
>>> card.unmatched()
True
>>> card.match(15)
True
>>> card.unmatched()
False
>>> card.reset()
>>> card.unmatched()
True
```

I dette eksempel oprettes en bingo plade med tallene 15, 27 og 73. Når man matcher tallet 27, returneres `True`, fordi tallet er på kortet. Dernæst findes 53 ikke på kortet, og 73 findes på kortet. Metoden tjekker, om der stadig er tal, der ikke er matchet, og returnerer `True`, fordi 15 stadig ikke er matchet. Derefter matches 15 efterfulgt af `match`, som returnerer `False`, da alle tal er blevet matchet. Kortet nulstilles, og nu er der nogle tal, der ikke er matchet.

Filnavnet og kravene er:

bingo_card.py

`BingoCard()`

A class representing a bingo card.

`__init__(numbers)`

Initializes the card with the given numbers.

Parameters:

- `numbers` `list` The numbers on the card.

`match(number)`

Match the number with the card.

Parameters:

- `number` `int` Number to match.

Returns:

- `bool` The number was matched.

`unmatched()`

Check whether any unmatched numbers are left on the card.

Returns:

- `bool` There are unmatched numbers.

`reset()`

Resets the card.

Opgave 10: Informativt bingokort (Informative Bingo Card)

Vi vil gerne lave en underklasse af `BingoCard`-klassen fra opgave 9, som giver mere informativ feedback.

Metoden `unmatched` skal returnere, hvor mange numre på kortet, der endnu ikke er blevet matchet.

Skriv klassedefinitionen for underklassen `InformativeBingoCard`, som arver fra `BingoCard`. Ændr de nødvendige metoder for at indarbejde denne ekstra adfærd, og arv de uændrede metoder fra moderklassen.

Du skal skrive klassedefinitionen for `InformativeBingoCard` i den samme fil som klassedefinitionen for `BingoCard`.

Se eksemplet nedenfor for den forventede opførsel.

```
>>> card = InformativeBingoCard([15, 27, 73])
>>> card.match(27)
True
>>> card.match(53)
False
>>> card.match(73)
True
>>> card.unmatched()
1
>>> card.match(15)
True
>>> card.unmatched()
0
>>> card.reset()
>>> card.unmatched()
3
```

I dette eksempel oprettes en informativ bingoplade med tallene 15, 27 og 73. Når man matcher tallet 27, returneres `True`, fordi tallet er på kortet. Dernæst findes 53 ikke på kortet, og 73 findes på kortet. Når man kontrollerer, om der stadig er tal, der ikke er matchet, returnerer metoden `1`, fordi 15 stadig ikke er matchet. Derefter matches 15, og når man tjekker, om der er tal, der ikke er matchet, returnerer metoden `0`. Kortet nulstilles, og nu er alle tre tal umatched.

Filnavnet og kravene er:

bingo_card.py

`InformativeBingoCard()`

A class representing a bingo card that provides more information.

`unmatched()`

Check how many numbers are unmatched.

Returns:

- `int` The number of unmatched numbers.