

Eksamen efterår 2024

Kursets navn:	Programmering
Kursusnummer:	02002 og 02003
Eksamensdato:	20. december 2024
Hjælpemidler tilladt:	Alle hjælpemidler, intet internet
Eksamens varighed:	4 timer
Vægtning:	Alle opgaver har samme vægt
Antal opgaver:	10
Antal sider:	12

Contents

- Eksamensinstruktioner
- Opgave 1: Drift af trafikken (Traffic Operation)
- Opgave 2: Omløbstid (Orbital Period)
- Opgave 3: Kammertryk (Chamber Pressure)
- Opgave 4: Niveauer i reservoir (Reservoir Levels)
- Opgave 5: Ny pris (New Price)
- Opgave 6: Frugtvægte (Fruit Weights)
- Opgave 7: Vægtet gennemsnit (Weighted Average)
- Opgave 8: Optælling af mønstre (Pattern Count)
- Opgave 9: Simple tracker (Simple Tracker)
- Opgave 10: Avanceret tracker (Advanced Tracker)

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 `2024_12_exam_English.pdf` og det samme dokument på dansk `2024_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 `2024_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 `2024_12_exam`. Hvis du bruger VS Code, skal du derfor klikke på `File` → `Open Folder...` og vælge mappen `2024_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.

- `chamber_pressure.py`
- `fruit_weights.py`
- `new_price.py`
- `orbital_period.py`
- `pattern_count.py`
- `reservoir_levels.py`
- `simple_tracker.py`
- `traffic_operation.py`
- `weighted_average.py`

Opgave 1: Drift af trafikken (Traffic Operation)

Trafiklys fungerer i en af følgende tre tilstande afhængigt af tidspunktet på dagen:

- *rush hour* fra 7:00-8:59 og 15:00-16:59,
- *night* fra kl. 22.00 til kl. 5.59 næste morgen,
- *normal* på alle andre tidspunkter.

Da driftstilstanden altid ændres i begyndelsen af en time, er den udelukkende en funktion af den aktuelle time (et tal fra 0 til 23).

Skriv en funktion, der tager den aktuelle time som input. Funktionen skal returnere driftstilstanden som en streng, enten

'normal', 'rush hour' eller 'night'.

Hvis den indtastede time f.eks. er 8, falder den inden for myldretiden mellem 7:00 og 8:59. Derfor skal funktionen returnere

'rush hour', som vist nedenfor.

```
>>> traffic_operation(8)
'rush hour'
```

Filnavnet og kravene er:

traffic_operation.py

`traffic_operation(hour)`

Determines the traffic light operation mode depending on the hour of the day.

Parameters:

- `hour` `int` The hour of the day.

Returns:

- `str` The traffic operation mode for the hour of the day.

Opgave 2: Omløbstid (Orbital Period)

Keplers tredje lov om planetbevægelser siger

$$T^2 = \frac{4\pi^2}{GM}a^3$$

hvor T er planetens omløbstid (i sekunder), a er længden af banens halve storakse (i meter), M er systemets samlede masse (i kilogram), og $G = 6.6743 \cdot 10^{-11} \text{ m}^3 \text{ kg}^{-1} \text{ s}^{-2}$ er gravitationskonstanten.

Skriv en funktion, der tager to tal som input: længden af den halve hovedakse (i meter) og systemets samlede masse (i kilogram). Funktionen skal returnere omløbstiden (i sekunder).

Som et eksempel kan vi betragte et system bestående af solen og jorden. Den halve storakse i Jordens bane er $1.5 \cdot 10^{11}$ meter, og systemets samlede masse er $2 \cdot 10^{30}$ kilogram. Vi har (uden at skrive enheder og kun med få decimaler)

$$T^2 = \frac{4\pi^2}{6.6743 \cdot 10^{-11} \cdot 2 \cdot 10^{30}} (1.5 \cdot 10^{11})^3 = 9.981 \cdot 10^{14}$$

så $T = \sqrt{9.981 \cdot 10^{14}} = 3.159 \cdot 10^7$ sekunder (hvilket er ca. 365.7 dage). Dette eksempel er vist nedenfor.

```
>>> orbital_period(1.5 * 10**11, 2.0 * 10**30)
31593584.1373
```

Filnavnet og kravene er:

orbital_period.py

`orbital_period(a, M)`

Calculate the orbital period of a planet using Kepler's third law.

Parameters:

- `a` `float` The length of the semi-major axis of the orbit, in meters.
- `M` `float` The total mass of the system, in kilograms.

Returns:

- `float` The orbital period of the planet, in seconds.

Opgave 3: Kammertryk (Chamber Pressure)

Trykket i et kammer stiger hver time med $k(P_{\max} - P)$, hvor P er det aktuelle tryk, $P_{\max}$ er det maksimale tryk, og k er en konstant. Vi er interesserede i den tid, det tager for trykket at nå eller overskride et kritisk tryk P_{crit} .

Skriv en funktion, der som input tager begyndelsestrykket, det maksimale tryk, det kritiske tryk og konstanten k . Funktionen skal returnere det antal timer, det tager for trykket at nå eller overskride det kritiske tryk.

Antag for eksempel, at begyndelsestrykket er $P = 20$, det maksimale tryk er $P_{\max} = 120$, det kritiske tryk er $P_{\text{crit}} = 105$, og konstanten er $k = 0.1$. Efter en time stiger trykket med $0.1(120 - 20) = 10$, hvilket bringer det op på 30. I den anden time stiger det med $0.1(120 - 30) = 9$, hvilket resulterer i et tryk på 39. Ved at gentage denne proces når trykket 104.99 efter 18 timer, stadig under den kritiske værdi. Efter 19 timer er trykket 106.49, hvilket overskrider den kritiske værdi. Dette er vist i koden nedenfor.

```
>>> chamber_pressure(20.0, 120.0, 105.0, 0.1)
19
```

Filnavnet og kravene er:

chamber_pressure.py

```
chamber_pressure(P0, Pmax, Pcrit, k)
```

Computes the number of hours it takes for the pressure to reach or exceed the critical value.

Parameters:

- `P0` `float` The initial pressure.
- `Pmax` `float` The maximum pressure.
- `Pcrit` `float` The critical pressure.
- `k` `float` The constant.

Returns:

- `int` The number of hours it takes for the pressure to reach or exceed the critical value.

Opgave 4: Niveauer i reservoir (Reservoir Levels)

Vandstanden i et reservoir registreres hver time. Hvis vandstanden falder med mere end 150 cm på en time, skal det rapporteres.

Skriv en funktion, der tager en liste af vandstande som input og returnerer en liste af indekser, hvor vandstanden falder mere end 150 cm i forhold til det forrige niveau.

Betragt for eksempel vandstandene `[1320, 1307, 1295, 1102, 1360, 1395, 1101, 1208]`. Niveauet ved indeks 0 har ikke noget tidligere niveau at sammenligne med. De næste to niveauer har fald på 13 og 12 cm. Niveauet ved indeks 3 er $1295 - 1102 = 193$ cm lavere end det foregående niveau, så dette er det første indeks, der skal rapporteres. De næste to niveauer er højere end de foregående niveau. Niveauet med indeks 6 falder med 294 cm i forhold til det forrige niveau, så det rapporteres. Det sidste niveau er højere end det forrige. Funktionen bør returnere `[3, 6]`, som vist nedenfor.

```
>>> reservoir_levels([1320, 1307, 1295, 1102, 1360, 1395, 1101, 1208])
[3, 6]
```

Filnavnet og kravene er:

reservoir_levels.py

`reservoir_levels(levels)`

Detects indices for water levels that are more than 150 lower than the previous measurement.

Parameters:

- `levels` `list` The list of water level measurements.

Returns:

- `list` The indices for water levels that are more than 150 lower than the previous measurement.

Opgave 5: Ny pris (New Price)

Når et produkt kommer på udsalg, reduceres prisen med en bestemt procentdel. Den nedsatte pris skal vises i et standardiseret format.

Hvis den oprindelige pris f.eks. er 143.50 DKK, og rabatten er 40%, beregnes den nye pris som

$$143.5 - 143.5 \frac{40}{100} = 86.1.$$

For at standardisere visningen afrunder vi den nye pris til præcis to decimaler, tilføjer et enkelt mellemrum og tilføjer valutakoden DKK. Prisstrengen bliver til `'86.10 DKK'`.

Skriv en funktion, der tager den oprindelige pris og rabatprocenten som input. Funktionen skal returnere en tuple, der indeholder den nye pris og den formaterede prisstreng, som vist i koden nedenfor.

```
>>> new_price(143.50, 40)
(86.1, '86.10 DKK')
```

Filnavnet og kravene er:

new_price.py

```
new_price(price, discount)
```

Computes the new price and formats the price string.

Parameters:

- `price` `float` The original price.
- `discount` `int` The discount percentage.

Returns:

- `tuple` The new price and the price string.

Opgave 6: Frugtvægte (Fruit Weights)

Du har en tabel med frugtvægte (i gram) for forskellige frugter, men frugtnavnene er ikke formateret konsekvent og vises med små bogstaver (*apple*), store bogstaver (*APPLE*) eller stort begyndelsesbogstav (*Apple*). Du vil gerne standardisere tabellen, så den kun indeholder navne med små bogstaver. Hvis der findes flere poster for den samme frugt, skal vægtene beregnes som et gennemsnit og rundes ned.

Skriv en funktion, der tager en tabel som input og returnerer en standardiseret tabel. Begge tabeller er *dictionaries*. Se f.eks. på nedenstående tabel.

```
>>> table = {'apple': 182,
...         'banana': 110,
...         'Orange': 160,
...         'Banana': 115,
...         'APPLE': 185,
...         'Apple': 175,
...         'lime': 67}
```

I denne tabel optræder *apple* i tre former med vægtene 182, 185 og 175 gram. I den standardiserede tabel skal navnet være `'apple'`, og vægten er gennemsnittet $(182+185+175)/3$ nedrundet til 180. *Banana* optræder i to former med vægten 110 og 115 gram. Så den standardiserede tabel skal have `'banana'` med vægten $(110+115)/2$ rundet ned til 112. *Orange* og *lime* optræder hver især én gang, så vægten forbliver uændret, men navnene skal konverteres til små bogstaver.

Du kan se det forventede output i eksemplet.

```
>>> fruit_weights(table)
{'apple': 180, 'banana': 112, 'orange': 160, 'lime': 67}
```

Filnavnet og kravene er:

fruit_weights.py

`fruit_weights(table)`

Standardizes table for fruit weights.

Parameters:

- `table` `dict` A dictionary of fruit names and weights.

Returns:

- `dict` A dictionary of fruit names and standardized weights.

Opgave 7: Vægtet gennemsnit (Weighted Average)

Givet N værdier $x_0, x_1, \dots, x_{N-1}$, vil du beregne deres vægtede gennemsnit. Vægtene bestemmes ud fra værdierne som

$$w_i = e^{\frac{-(x_i - \mu)^2}{2\sigma^2}}$$

hvor μ er gennemsnittet af værdierne, og σ er standardafvigelsen. Gennemsnittet og standardafvigelsen kan beregnes som

$$\mu = \frac{1}{N} \sum_{i=0}^{N-1} x_i \quad \text{og} \quad \sigma = \sqrt{\frac{1}{N} \sum_{i=0}^{N-1} (x_i - \mu)^2}.$$

Det vægtede gennemsnit beregnes derefter som

$$a = \frac{\sum_{i=0}^{N-1} w_i x_i}{\sum_{i=0}^{N-1} w_i}.$$

Betrakt for eksempel værdierne $[4.8, 6.6, 12.2, 7.3, 6.5]$. Gennemsnittet og standardafvigelsen er henholdsvis

$\mu = 7.48$ og $\sigma = 2.499$. Vægten for det første element er $e^{\frac{-(4.8-7.48)^2}{2 \cdot 2.499^2}} = 0.563$, og alle vægte er $[0.563, 0.94, 0.168, 0.997, 0.926]$. Endelig er det vægtede gennemsnit

$$a = \frac{0.563 \cdot 4.8 + \dots}{0.563 + \dots} = \frac{24.254}{3.594} = 6.749.$$

I et særligt tilfælde, hvor $\sigma = 0$, f.eks. hvis alle værdier x_i er ens, er det vægtede gennemsnit lig med gennemsnittet af værdierne.

Skriv en funktion, der tager et NumPy-array med værdier som input og returnerer det vægtede gennemsnit. Du kan se det forventede output i eksemplet.

```
>>> import numpy as np
>>> weighted_average(np.array([4.8, 6.6, 12.2, 7.3, 6.5]))
np.float64(6.748513328262833)
```

Filnavnet og kravene er:

weighted_average.py

`weighted_average(x)`

Computes the weighted average of the array.

Parameters:

- `x` `numpy.ndarray` An array of numbers.

Returns:

- `float` The weighted average of the array.

Opgave 8: Optælling af mønstre (Pattern Count)

En DNA-streng er repræsenteret som en tekst, der indeholder bogstaverne `A`, `C`, `G` og `T`, og den er gemt i en fil. Målet er at tælle, hvor mange gange et givet mønster optræder i DNA-strengen. Der kan være linjeskift i filen, men de har ingen betydning og skal ignoreres. Alle forekomster af mønsteret skal tælles, også selvom de overlapper hinanden. For eksempel optræder mønsteret `ACA` to gange i DNA-strengen `ACACA`.

Skriv en funktion, der tager et filnavn og et mønster som input og returnerer det antal gange, mønsteret optræder i DNA-strengen.

Tænk for eksempel på en fil `files/dna_data_1.txt` med følgende indhold:

```
CGGTGCGGGCCTCTTCGCTATTACGCCAGCTGGCG
AAAGGGGGATGTGCTGCAAGGCGATTAAGTTGGGTAACGCCAGGG
TTTTCCAGTCACGACGTTGTAAAACGACGGCCAGT
GAGCGCGCGTAATACGACTCACTATAGGGCGAATTGGAGCTCCA
CCGCGGTGGCGGCCGCTCTAGAAGTGGATCCCCCGG
GCTGCAGGAATTCGATATCAAGCTTATCGATACCGTCGACC
TCGAGGGGGGGCCCGGTACCCAGCTTTTGTCCCTTTAGTGAG
GGTTAATTGCGCGCTTG
```

og et mønster `'ACCG'`. Dette mønster optræder to gange i DNA-strengen: en gang med `A` i slutningen af den fjerde linje, som fortsætter i den femte linje, og en anden gang med `A`, som er placeret 10 tegn før slutningen af den sjette linje.

Det forventede output kan ses i eksemplet.

```
>>> filename = 'files/dna_data_1.txt'
>>> pattern_count(filename, 'ACCG')
2
```

Filnavnet og kravene er:

`pattern_count.py`

`pattern_count(filename, pattern)`

Finds the number of occurrences of the pattern in the DNA strand.

Parameters:

- `filename` `str` The name of the file with the DNA strand.
- `pattern` `str` The pattern to search for.

Returns:

- `int` The number of occurrences of the pattern in the DNA strand.

Opgave 9: Simple tracker (Simple Tracker)

Vi vil lave en tracker, der kan holde styr på flere tal, men kun dem, der er positive, og som ikke overskrider en bestemt *limit*. Trackeren skal starte tom, og tal tilføjes til trackeren et efter et, hvis de opfylder kravene. Trackeren skal være i stand til at beregne simpel statistik over de tilføjede tal. Det skal være muligt at nulstille trackeren.

Skriv en klassedefinition for klassen `SimpleTracker`. Konstruktøren skal tage et positivt heltal, *limit*. Trackeren skal være tom til at begynde med. Metoden `add` skal tage et tal som input og tilføje det til de tilføjede tal, hvis det er positivt og ikke større end *limit*. Metoden `add` skal returnere `True`, hvis tallet blev tilføjet, og `False`, hvis det ikke blev tilføjet. Metoden `reset` skal slette de tilføjede tal. Metoden `stats` skal returnere en tupel med to elementer: *total* og *span*. Her er *total* summen af alle tilføjede tal, og *span* er forskellen mellem det største og det mindste tilføjede tal. Hvis trackeren er tom, skal både *total* og *span* være 0.

Se på eksemplet nedenfor.

```
>>> tracker = SimpleTracker(500)
>>> tracker.add(510)
False
>>> tracker.add(200)
True
>>> tracker.add(352)
True
>>> tracker.stats()
(552, 152)
>>> tracker.reset()
>>> tracker.stats()
(0, 0)
```

I dette eksempel er der oprettet en tracker med en *limit* på 500. Når man forsøger at tilføje tallet 510, returneres `False`, fordi tallet overskrider *limit* og ikke kan tilføjes til trackeren. Dernæst tilføjes tallene 200 og 352 med succes. På dette tidspunkt er *total* og *span* $200 + 352 = 552$ og $352 - 200 = 152$. Derefter nulstilles trackeren, og både *total* og *span* er 0.

Filnavnet og kravene er:

simple_tracker.py

`SimpleTracker()`

A class that tracks valid numbers.

`__init__(limit)`

Initialize a tracker with a limit.

Parameters:

- `limit` `int` The limit that valid numbers cannot exceed.

`add(number)`

Add a valid number to the tracked numbers.

Parameters:

- `number` `int` The number to be added.

Returns:

- `bool` `True` if the number is added, `False` otherwise.

`reset()`

Reset tracker.

`stats()`

Return the total and the span.

Returns:

- `tuple` The total and the span of the tracked numbers.

Opgave 10: Avanceret tracker (Advanced Tracker)

Vi vil lave en underklasse af `SimpleTracker`-klassen fra opgave 9, som udfører en ekstra kontrol, før den tilføjer et nummer til trackeren.

Når man forsøger at tilføje et nyt tal, hvis trackeren ikke er tom, skal det nye tal kun tilføjes, hvis det ikke adskiller sig fra det forrige med mere end en bestemt værdi, *max delta*. Hvis den absolutte forskel overstiger *max delta*, skal tallet ikke tilføjes. Derudover skal tallet stadig kontrolleres for gyldighed som i den overordnede klasse, og metoden `add` skal returnere `True`, hvis tallet blev tilføjet, og `False`, hvis det ikke blev det.

Skriv klassedefinitionen for underklassen `AdvancedTracker`, som arver fra `SimpleTracker`. Konstruktøren skal tage både *limit* og *max delta* som inputparametre. Ændr de nødvendige metoder for at indarbejde denne ekstra adfærd, og arv de uændrede metoder fra moderklassen.

Du skal skrive klassedefinitionen for `AdvancedTracker` i den samme fil som klassedefinitionen for `SimpleTracker`.

Se eksemplet nedenfor for forventet adfærd.

```
>>> tracker = AdvancedTracker(500, 100)
>>> tracker.add(200)
True
>>> tracker.add(352)
False
>>> tracker.add(252)
True
>>> tracker.stats()
(452, 52)
>>> tracker.add(510)
False
```

I dette eksempel initialiseres trackeren med en *limit* på 500 og en *max delta* på 100. Tallet 200 tilføjes med succes. Tallet 325 tilføjes ikke, fordi den absolutte forskel mellem dets værdi og den foregående $352-200=152$ er større end *max delta*, som er 100. Tallet 252 tilføjes med succes, fordi den absolutte forskel mellem dets værdi og den foregående er 52. På dette tidspunkt er *total* $252+200=452$, og *span* er $252-200=52$. Når man forsøger at tilføje 510, bliver det ikke tilføjet, fordi det overskrider *limit* på 500.

Filnavnet og kravene er:

simple_tracker.py

`AdvancedTracker()`

A class that tracks of non-extreme valid numbers.

`__init__(limit, max_delta)`

Initialize a record with a limit and a maximum delta.

Parameters:

- `limit` `int` The limit that valid numbers cannot exceed.
- `max_delta` `int` The maximum delta value for determining non-extreme numbers.

`add(number)`

Add a valid and non-extreme number to the tracked numbers.

Parameters:

- `number` `int` The number to be added.

Returns:

- `bool` `True` if the number is added, `False` otherwise.