

Exam 2024 May

Course name:	Computer programming
Course number:	02002 and 02003
Exam date:	30th of May 2024
Aids allowed:	All aids, no internet
Exam duration:	4 hours
Weighting:	All tasks have equal weight
Number of tasks:	10
Number of pages:	13

Contents

- Exam Instructions
- Task 1: Distance Traveled
- Task 2: Booklet Layout
- Task 3: Reversed Text
- Task 4: First Double Peak
- Task 5: Robust Values
- Task 6: Population Convergence
- Task 7: Event Manager
- Task 8: Name Frequency
- Task 9: Count Differences
- Task 10: Limited Event Manager

Exam Instructions

Exam Material

The exam material is a single zip file, which you should unzip to a folder on your computer. The zip file contains the exam text as a PDF document in English `2024_05_exam_English.pdf` (this document) and the same document in Danish `2024_05_exam_Danish.pdf`. Each pdf contains everything needed to solve the exam.

As additional help, the zip file contains a folder `2024_05_exam` with the following content:

- Empty Python files `<task_name>.py`, where `<task_name>` is the name of the task. There may be fewer files than tasks.
- A Python test file for each task, `test_task_<n>_<task_name>.py`, where `<n>` is the task number, and `<task_name>` is the name of the task.
- A Python file `test_tasks_all.py`.
- A folder `files` containing data files, if there are any.

Solving Exam Tasks

To be able to solve the exam tasks, you need to have a computer with Python installed. All tasks can be solved using any editor of your choice, such as IDLE or VS Code.

We recommend that you use the provided files: Write your solutions in the empty files and test them by running the provided test scripts. These scripts check whether your solution behaves correctly for the example in the task description. Use `test_tasks_all.py` to run all tests at once. The provided test scripts assume that the *current working directory* is `2024_05_exam`. Therefore, if you are using VS Code, you should click `File` → `Open Folder...` and select the `2024_05_exam` folder (which is *inside* the folder you unzipped).

A solution that fails the provided test is incorrect. If your solution passes the provided test, it is not a guarantee that your solution is correct.

Your solutions may only use the tools taught in the course. Solutions that import modules other than `math`, `numpy`, `os`, or `matplotlib` will not be graded. The provided test scripts do not check for this, so it is your responsibility to ensure that your solutions only use the allowed modules.

If you believe there is a mistake or ambiguity in the text, use the most reasonable interpretation of the text and solve the task to the best of your ability. If we, after the exam, find inconsistencies in one or more tasks, this will be taken into account during the assessment.

Evaluation of the Exam

We will run additional tests for each task to check whether your solutions behave as specified in the task. The fraction of passed tests is the score for each task. The overall score is the average of the task scores.

Handing in

To hand in your solutions, upload your Python files with solutions to the Digital Exam system. In the Digital Exam system, files can be submitted either as the *main document* or as *attachments*. You may upload *any* one of your solution files as the main document, and the remaining files as attachments.

Please submit only the following files with these exact names:

- `booklet_layout.py`
- `count_differences.py`
- `distance_traveled.py`
- `event_manager.py`
- `first_double_peak.py`
- `name_frequency.py`
- `population_convergence.py`
- `reversed_text.py`
- `robust_values.py`

Any file handed in that is not in the list above will not be taken into account in your assessment.

Task 1: Distance Traveled

The distance traveled by an object falling from standstill is calculated using the formula

$$s = \frac{1}{2}gt^2,$$

where s is the distance traveled (in meters), t is the duration of the fall (in seconds), and g is the gravitational acceleration on Earth, equal to 9.81m/s^2 .

You should write a function that takes as input the duration of the fall (in seconds). The function should return the distance traveled (in meters).

As an example, consider an object falling for 5.5 seconds. The distance traveled is

$$\frac{1}{2} 9.81 \cdot 5.5^2$$

which is what your function should return, as shown in the code cell below.

```
>>> distance_traveled(5.5)
148.37625
```

The filename and requirements are in the box below:

distance_traveled.py

`distance_traveled(t)`

Return the distance traveled by the object falling for t seconds.

Parameters:

- `t` `float` A non-negative float, the duration of the fall in seconds.

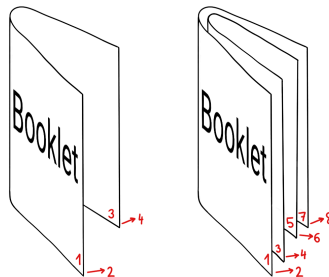
Returns:

- `float` The distance traveled in meters.

Task 2: Booklet Layout

A booklet may be made by folding sheets of paper, as in the illustration below. When only one sheet of paper is used, the booklet has 4 pages. If two sheets are used, the booklet has 8 pages. Every additional sheet contributes with 4 pages. Therefore, the number of pages in a booklet is always a multiple of 4.

If we have a certain number of pages with content, and this number is not a multiple of 4, there will be up to 3 blank pages at the end of the booklet.



Given a number of pages with content to be placed in a booklet, we want to know two things:

- the total number of pages in the smallest booklet that can accommodate the content,
- the number of blank pages in such a booklet.

Write a function that takes as input the number of pages of content. The function should return the total number of pages in the smallest appropriate booklet, and the number of blank pages.

As an example, consider having 17 pages with content. Number 17 is not a multiple of 4, so pages need to be added. Adding one or two blank pages will not be enough, since neither 18 nor 19 are multiples of 4. Adding three blank pages will give 20, which is a multiple of 4. Therefore, the booklet has 20 pages, and there will be 3 blank pages. The desired output is shown in the code cell below.

```
>>> booklet_layout(17)
(20, 3)
```

The filename and requirements are in the box below:

booklet_layout.py

`booklet_layout(content_pages)`

Return the number of total and blank pages given content.

Parameters:

- `content_pages` `int` A positive integer, the number of pages with content.

Returns:

- `tuple` The number of total pages and the number of blank pages.

Task 3: Reversed Text

Given a string of text, we want to reverse either the order of the words in the text, or the order of the letters in each word. You can assume that the string of text only contains words written with letters from the English alphabet, and that the words are separated by a single space.

Write a function that takes as input a string of words and a string with the option `words` or `letters`. The function should return the string of text with the specified reversal. If the option is `words`, the order of the words should be reversed. If the option is `letters`, the order of the letters in each word should be reversed.

Consider the example below.

```
>>> reversed_text('Hello world we are going to do some programming', 'letters')
'olleH dlrow ew era gniog ot od emos gnimmargorp'
```

In this example, the option is `letters`, so the order of the letters in each word is reversed.

The filename and requirements are in the box below:

`reversed_text.py`

`reversed_text(text, option)`

Reverse words or letters.

Parameters:

- `text` `str` A text consisting of words separated by spaces.
- `option` `str` A string, either 'words' or 'letters'.

Returns:

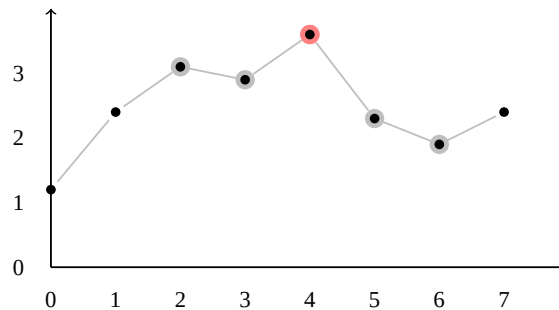
- `str` The reversed text.

Task 4: First Double Peak

Given a list of numbers, we want to locate the first peak. Usually, a peak is a number that is strictly larger than its first neighbors (the number just before and the number just after). However, in this task, we want to locate a double peak, which is a number that is strictly larger than both its first and its second neighbors (the two numbers before and the two numbers after).

Write a function that takes as input a list of floats. The function should return the index of the first double peak. If there is no double peak, the function should return **-1**.

As an example, consider the list `[1.2, 2.4, 3.1, 2.9, 3.6, 2.3, 1.9, 2.4]`. The numbers from the list are also in the figure below, where the x-axis represents the index of the numbers and the y-axis represents the values of the numbers.



Considering all numbers in order, the first two values should be ignored, as they have no two neighbors before. The value 3.1 is not strictly larger than its second neighbor with the value 3.6. The value 2.9 is not a peak either, as it is not strictly larger than 3.1. The value 3.6 (red) is a double peak as it is larger than both 3.1, 2.9, 2.3 and 1.9 (gray). The function should therefore return the index of the value 3.6, which is **4**, as shown in the code cell below.

```
>>> first_double_peak([1.2, 2.4, 3.1, 2.9, 3.6, 2.3, 1.9, 2.4])
4
```

The filename and requirements are in the box below:

`first_double_peak.py`

`first_double_peak(sequence)`

Return first number strictly larger than its first and second neighbors.

Parameters:

- `sequence` `list` A list of floats.

Returns:

- `int` The index of the first peak.

Task 5: Robust Values

Given a NumPy array of numbers, we want find the values which are not more than one standard deviation away from the mean.

Given N numbers x_i , the mean and the standard deviation are

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

The robust values (that we want to keep) are less than exactly one standard deviation away from the mean, i.e. a robust value x_i satisfies $\mu - \sigma \leq x_i$ and $x_i \leq \mu + \sigma$.

Write a function which takes as input a NumPy array. The function should return a NumPy array containing only the robust values in the same order as in the original array.

As an example, consider the input below.

```
>>> import numpy as np
>>> x = np.array([41.42, 44.32, 45.56, 63.01, 12.22, 42.82, 43.73, 40.11])
```

The mean of the numbers is $\mu = 41.65$, and the standard deviation is $\sigma = 13.00$ (all values are here displayed with two decimals). The robust values are in the interval $[28.64, 54.65]$, so only values 63.01 and 12.22 should be removed, as seen in the code cell below.

```
>>> robust_values(x)
array([41.42, 44.32, 45.56, 42.82, 43.73, 40.11])
```

The filename and requirements are in the box below:

robust_values.py

`robust_values(x)`

Return values within one standard deviation from the mean of the input.

Parameters:

- `x` `numpy.ndarray` A NumPy array.

Returns:

- `numpy.ndarray` A NumPy array with robust values.

Task 6: Population Convergence

We investigate a model which describes the change in the size of the population from year to year. The model is given by the formula

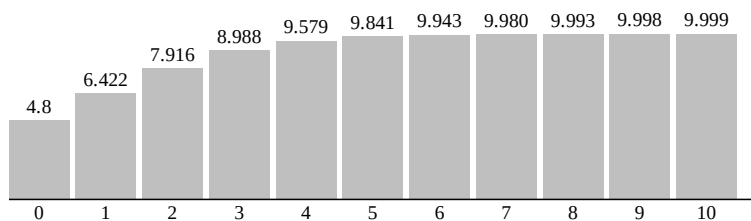
$$N_{t+1} = N_t + r \left(1 - \frac{N_t}{K} \right) N_t$$

where N_t is the population size (in thousands) at year t . The model parameters are the growth rate r and the carrying capacity K . According to this model, if the parameters are reasonable (and you can assume this is the case), the population will converge to the carrying capacity K : if the population is larger than K , it will decrease, and if it is smaller than K , it will increase.

We only investigate the situation where $K = 10$ and we want to know how many years it takes for the population to be strictly within 1% of K . In other words, we want to know in how many years will the population size be in the interval between 9.9 and 10.1, not including the limits.

Write a function that takes as input the initial population size and the growth rate. The function should return the number of years it takes for the population to be strictly within 1% of $K = 10$.

Consider the growth rate $r = 0.65$ and initial population $N_0 = 4.8$. The initial population is not within the interval between 9.9 and 10.1, so we compute the population after one year. We have $N_1 = 4.8 + 0.65 \cdot (1 - 4.8/10) \cdot 4.8 = 6.422$ (the printed value is rounded) which is also not within the interval. The population growth continues as shown in the figure below, where the x-axis represents the years and the y-axis represents the population size. The population size for each year is shown with three decimals.



After 6 years, the (rounded) population size is 9.943, which is in the interval within 1% of K , so the function should return 6, as shown in the code cell below.

```
>>> population_convergence(4.8, 0.65)
6
```

The filename and requirements are in the box below:

population_convergence.py

```
population_convergence(N, r)
```

Return the number of years for population to be within 1% of $K=10$.

Parameters:

- **N** `float` A positive float, the initial population.
- **r** `float` A positive float, the growth rate.

Returns:

- `int` The number of years.

Task 7: Event Manager

We want to create a class to represent an event (like a lecture or a concert), allowing for registering and de-registering participants, while preventing duplicate registrations.

Write the class definition for the class `EventManager`. The `register` method should take a name as input and add it to the list of registrations. If the name is already in the list, it should not be added again. The method should return `True` if the name was added, and `False` if it was not. The `deregister` method should take a name as input, remove it from the list of registrations and return `True`. If the name is not in the list it cannot be removed, and `False` should be returned. The `get_num_registrations` method should return the number of participants currently registered.

Below is an example of using the class.

```
>>> my_event = EventManager()
>>> my_event.get_num_registrations()
0
>>> my_event.deregister('Mike')
False
>>> my_event.register('Mike')
True
>>> my_event.register('Mike')
False
>>> my_event.register('John')
True
>>> my_event.deregister('Mike')
True
>>> my_event.get_num_registrations()
1
```

In this example, there are no registrations initially. Then, an attempt to deregister **Mike** is made, but this is not possible. Then, **Mike** is registered. Then, an attempt to register **Mike** again is made, but this is not possible. Then, **John** is registered. Finally, **Mike** is deregistered. Finally, the number of registrations is checked and printed.

The filename and requirements are in the box below:

event_manager.py

`EventManager()`

A class that represents an event.

`__init__()`

Initialize the event with no registrations.

`register(user)`

Register the user as an event participant.

Parameters:

- `user` `str` The user name to register.

Returns:

- `bool` True if the user was successfully registered, False otherwise.

`deregister(user)`

Deregister the user.

Parameters:

- `user` `str` The user name to deregister.

Returns:

- `bool` True if the user was successfully deregistered, False otherwise.

`get_num_registrations()`

Return the number of users currently registered.

Returns:

- `int` The number of registered users.

Task 8: Name Frequency

Given a list of full names, we need to know how many times each first name occurs in the list. Here, the first name is the part of the full name before the first space.

Write a function that takes a list of full names as input. The function should return a dictionary where the keys are the first names from the list. The value of each key should be the number of times this first name occurs in the list.

As an example, consider the input below.

```
>>> names = ['Liv Ea Jensen',
...          'Mads Oliver',
...          'Steve Madsen',
...          'Anna Simon',
...          'Simon Gade',
...          'Mads Kai Jensen']
```

The first names are **Liv**, **Mads**, **Steve**, **Anna**, **Simon**, and **Mads**. The first name **Mads** occurs twice, and the other first names occur once. The function should therefore return the dictionary with keys and values as shown below.

```
>>> name_frequency(names)
{'Liv': 1, 'Mads': 2, 'Steve': 1, 'Anna': 1, 'Simon': 1}
```

The filename and requirements are in the box below:

name_frequency.py

`name_frequency(names)`

Return frequency of names in the list.

Parameters:

- `names` `list` A list of strings.

Returns:

- `dict` The frequency of names.

Task 9: Count Differences

The results of an experiment are recorded by two independent observers. The observers record the results as a sequence of comma-separated integers, which is saved in a file containing one line of text. We need to count the number of differences between the recorded results of the two observers.

Write a function that takes as input two strings containing the names of the files with the experiment results. If the number of results in one file is different from the number of results in the second file, the function should return **-1**. If the number of results is the same in the two files, the function should return the number of results that the two observers have recorded differently. Consequently, the function should return **0** if the results in both files are the same.

As an example, consider the two files below.

```
>>> filename1 = 'files/results_A1.txt'
>>> filename2 = 'files/results_A2.txt'
```

The content of the first file is:

```
345, 349, 367, 299, 345, 445, 345, 465, 299, 345
```

The content of the second file is:

```
345, 349, 367, 300, 354, 445, 345, 465, 300, 345
```

Both files contain 10 recorded results, so we inspect each pair of recorded results. The first three pairs are the same (345, 349, 367) but the fourth pair is different (299 and 300). Furthermore, the fifth and ninth pairs are different. The function should therefore return **3**, as shown in the code cell below.

```
>>> count_differences(filename1, filename2)
3
```

The filename and requirements are in the box below:

count_differences.py

```
count_differences(filename1, filename2)
```

Number of differences in recorded results.

Parameters:

- `filename1` `str` Filename of the first file.
- `filename2` `str` Filename of the second file.

Returns:

- `int` Number of differences in recorded results.

Task 10: Limited Event Manager

We want to create a subclass of the `EventManager` class from Task 7. This subclass should prevent the registration of participants if the limit on the number of participants has been reached.

Write the class definition for the subclass `LimitedEventManager`, which inherits from `EventManager`. Each instance of the subclass should store the registration limit. The constructor of the new class should take as input the registration limit (a non-negative integer). The `register` method should ensure that the number of registrations does not exceed the limit. If the registration is not possible because the limit is reached, the method should return `False`. If the limit is not reached, the registration method should act as specified in `EventManager`. You should modify the necessary methods of the class to achieve this behavior, and inherit the rest of the methods from the parent class.

An example of using the class is shown below.

```
>>> my_event = LimitedEventManager(3)
>>> my_event.register('Mike')
True
>>> my_event.register('Emily')
True
>>> my_event.register('Sara')
True
>>> my_event.register('Peter')
False
>>> my_event.get_num_registrations()
3
```

In this example, there are no registrations initially. Then, **Mike** is registered, and then **Emily** and **Sara** are registered. Then, an attempt to register **Peter** is made, but this is not possible as 3 people are already signed up. Finally, we get and print the number of registrations.

The filename and requirements are in the box below:

event_manager.py

`LimitedEventManager()`

A class that represents an event with limited number registrations.

`__init__(registration_limit)`

Initializes the limited event with a registration limit.

Parameters:

- `registration_limit` non-negative `int` The maximum number of allowed registrations.

`register(user)`

Register the user if limit is not reached.

Parameters:

- `user` `str` The user name to register.

Returns:

- `bool` True if the user was successfully registered, False otherwise.