

# Exam Summer 2026

|                         |                             |
|-------------------------|-----------------------------|
| <b>Course name:</b>     | Computer programming        |
| <b>Course number:</b>   | 02002 and 02003             |
| <b>Exam date:</b>       | 30th of May 2026            |
| <b>Aids allowed:</b>    | All aids, no internet       |
| <b>Exam duration:</b>   | 4 hours                     |
| <b>Weighting:</b>       | All tasks have equal weight |
| <b>Number of tasks:</b> | 10                          |
| <b>Number of pages:</b> | 12                          |

## Contents

- Exam Instructions
- Task 1: Pendulum Period
- Task 2: Count Negatives
- Task 3: Digit Count
- Task 4: Sequence Difference
- Task 5: Hour Point Distance
- Task 6: Buying Options
- Task 7: Password Analysis
- Task 8: Intensity Minutes
- Task 9: Card Player
- Task 10: Container

# 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 `2026_05_exam_English.pdf` (this document) and the same document in Danish `2026_05_exam_Danish.pdf`. Each pdf contains everything needed to solve the exam.

As additional help, the zip file contains a folder `2026_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 input 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 `2026_05_exam`. Therefore, if you are using VS Code, you should click `File` → `Open Folder...` and select the `2026_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.

All tasks can be solved using only 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, 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. All other files will be ignored.

- `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`

# Task 1: Pendulum Period

The period  $T$  (in seconds) of a pendulum is

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

where  $L$  is the length of the pendulum (in meters) and  $g = 9.81\text{m/s}^2$  is the acceleration due to gravity.

For example, a pendulum of length 1.5 meters has a period of  $2\pi\sqrt{\frac{1.5}{9.81}} = 2.4569$  seconds.

Write a function that takes the pendulum length in meters as input and returns its period in seconds.

The behavior of the function is shown below.

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

The filename and requirements are:

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.

## Task 2: Count Negatives

A list contains integers. Each integer may be positive, negative, or zero. We want to count how many integers in the list are negative. If the list is empty, the count is 0.

For example, consider the list

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

It contains in total five negative integers, namely -2, -10, -1, -1, and -2. Therefore, the count is 5.

Write a function that takes a list as input and returns the count.

The behavior of the function is shown below.

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

The filename and requirements are:

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.

## Task 3: Digit Count

Given a positive integer and a digit from 0 to 9, we want to count how many times the digit appears when you write all numbers from 1 up to the given integer.

For example, consider the number 13 and a digit 1. In the sequence

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

the digit 1 occurs in five numbers: 1, 10, 11, 12, and 13. It occurs twice in 11. In total, the digit 1 occurs 6 times.

Write a function that takes a positive integer and a digit as input. The function should return how many times the digit occurs in the sequence from 1 to the given number.

The behavior of the function is shown below.

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

The filename and requirements are:

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.

## Task 4: Sequence Difference

Given two sequences of numbers of the same length, we want to calculate the difference between the sequences. The difference  $d$  between sequences  $a_i$  and  $b_i$  defined as the sum of the absolute differences between the corresponding elements

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

For example, consider a sequence 1.2, 2.4, 3.6 and a sequence 4.2, 0.2, 6.6. The difference is

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

Write a function that takes as input two lists of numbers, each representing a sequence. The function should return the difference. You can assume that the two lists have the same length.

The behavior of the function is shown below.

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

The filename and requirements are:

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.

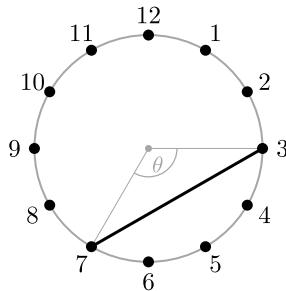
## Task 5: Hour Point Distance

A clock face has points numbered 1 to 12 on a circle of radius 1. We want to find the distance between any two of such hour-points.

One approach is this:

- Find the shortest number of steps between the two hour-points (clockwise or counterclockwise).
- Convert steps to an angle  $\theta$ , where each step is  $\frac{2\pi}{12}$  radians (one twelfth of the full circle).
- Use the formula for the distance between two points on the unit circle  $d = 2 \sin\left(\frac{\theta}{2}\right)$ .

For example, consider the hour-points 7 and 3, shown below.



Starting from 7, the shortest path to 3 is to move 4 steps counterclockwise. So, the angle between the two hour-points is  $\theta = 4 \cdot \frac{2\pi}{12}$  and the distance is

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

Write a function that takes as input two hour-points (integers from 1 to 12) and returns the distance between them.

The behavior of the function is shown below.

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

The filename and requirements are:

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.

## Task 6: Buying Options

A *dictionary* contains items and their price per unit. Given an amount, we want a *dictionary* with items and number of whole units that can be bought for that amount.

Write a function that takes a *dictionary* of prices and an amount as input. The function should return a *dictionary* with items and number of units that can be bought. Exclude items for which the amount is not enough to buy one whole unit. Do not modify the original *dictionary*.

For example, consider the *dictionary*

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

and the amount 15.0.

With this amount, you can buy 3 mangoes ( $15.0/4.5 = 3.33$  which is enough for 3 but not 4), or 4 bananas (since  $15.0/3.5 = 4.28$  rounds down to 4), or 1 chocolate ( $15.0/10.0 = 1.5$ , which rounds down to 1). The returned *dictionary* can be seen below.

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

The filename and requirements are:

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.



## Task 7: Password Analysis

A password consists of digits and letters from the English alphabet. We want to count the number of letters and digits in a password string.

For example, the password "PASS123word" has 8 letters (P, A, S, S, w, o, r, d) and 3 digits (1, 2, 3).

Write a function that takes a password string as input and returns a tuple containing the number of letters and the number of digits in the password. You can assume the password contains only English letters (A–Z, a–z) and digits (0–9).

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

The filename and requirements are:

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.

## Task 8: Intensity Minutes

A file contains daily activity records. Each line in the file contains two integer values: the number of minutes for an activity and its intensity level. The intensity level is 1 for low intensity, 2 for moderate intensity, and 3 for vigorous intensity. The total intensity minutes for a day is the sum of minutes from level 2 and level 3 activities, where minutes from level 3 activities count double.

Write a function that takes a filename as input and returns the total intensity minutes for the day.

For example, consider the following file:

files/activities\_1.txt

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

There are five activities in the file. Three activities are at level 2 with durations of 46, 21, and 3 minutes, contributing  $46 + 21 + 3 = 70$  intensity minutes. One activity is at level 3 with a duration of 8 minutes, contributing  $2 \cdot 8 = 16$  intensity minutes. The level 1 activity, with a duration of 15 minutes, does not contribute to the intensity minutes. Therefore, the total intensity minutes is  $70 + 16 = 86$ , as shown below.

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

The filename and requirements are:

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.

## Task 9: Card Player

A card player starts with 0 points and a hand of cards. Each card has a distinct value between 1 and 100. When an opponent plays a card, the player checks whether their hand contains one or more cards with a higher value. If so, the player uses the smallest such card and gains points equal to the sum of that card and the opponent's card. If not, the player uses the smallest card in their hand and gains no points.

Write a class `CardPlayer` with a constructor that takes a list of integers representing the player's hand. The class should have a method `play` that takes an integer that represents the card played by the opponent. The method should select the appropriate card to use, remove it from the hand, and return the player's total points after using the card. You can assume that all cards in the game (including opponent cards) have different values, and that `play` is only called when the player has at least one card in the hand.

The behavior is illustrated below for an initial hand with cards 42, 30, and 50. First, the opponent plays 35, and the player uses 42, gaining  $35 + 42 = 77$  points. Next, the opponent plays 61, and the player uses 30, gaining no additional points. Finally, the opponent plays 10, and the player uses 50, gaining  $10 + 50 = 60$  points. The total points is therefore  $77 + 60 = 137$ , as shown below.

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

The filename and requirements are:

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.

## Task 10: Container

A container has an integer capacity and holds an integer amount between 0 and the capacity.

Write a class `Container` with a constructor that takes the capacity as input, and creates an empty container. The method `fill` takes an integer amount to fill into the container. The method fills the container, but never exceeds its capacity. It returns the amount actually filled. The method `remove` takes an integer amount to remove from the container. The method removes from the container, but never below 0. It returns the amount actually removed. The method `__str__` returns a string representation of the container in the format `"Container: x/y"`, where `x` is the current amount in the container and `y` is the capacity.

The behavior is illustrated below with a container of capacity 10. First, fill 7. Then, attempt to fill 5 but only 3 fit. Finally, attempt removing 14 but only 10 are available.

```
>>> 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'
```

The filename and requirements are:

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"`.