# Technical University of Denmark

| | |
|---|---|
| Course name | Introduction to programming and data processing |
| Course number | 02633 (02631, 02632, 02634) |
| Aids allowed | All Aids |
| Exam duration | 2 hours |
| Weighting | All exercises have equal weight |

## Contents

## Submission details

You must hand in your solution electronically:

1. You can test your solutions individually on Code Judge under *Exam*. When you upload a solution on CodeJudge, the test example given in the assignment description will be run on your solution. If your solution passes this single test, it will appear as *Submitted*. This means that your solution passes on this single test example. You can upload to CodeJudge as many times as you like during the exam.

2. You must upload all your solutions at Online Exam. Each assignment must be uploaded as one separate .py file, given the same name as the function in the assignment:

   (a) `water_height.py`

   (b) `astronomical_season.py`

   (c) `tictactoe.py`

   (d) `standardize_address.py`

   (e) `time_angle.py`

   The files must be handed in separately and must have these exact filenames.

After the exam, your solutions submitted to Online Exam will be automatically evaluated on CodeJudge on a range of different tests, to check that they work correctly in general. The assessment of your solution is based only on how many of the automated tests it passes.

- Make sure that your code follows the specifications exactly.
- Each solution shall not contain any additional code beyond the specified function, though `import` statements can be included.
- Remember, you can check if your solutions follow the specifications by uploading them to CodeJudge.
- Note that all vectors and matrices used as input or output must be numpy arrays.

## Assignment 1 Water height

The height of the water in a pond changes daily governed by two contributions: the constant decrease of height due to the water outflow and the variable increase of height due to the rain. Given the water height for one day and the rain value $r$, the water height for the next day can be computed as

$$h_{\text{today}} = h_{\text{yesterday}} + r - 2.$$

If the computed value is negative it should be replaced by 0, indicating that the pond is empty. The computation can be repeated for any number of days, as long as we have rain values.

### Problem definition

Write a function `water_height` which takes an initial water height $h_0$ and a vector $\mathbf{r}$ with rain values for a number of days as input. The function should return a water height after the days have passed. The function should also work if $\mathbf{r}$ contains only one or no days. (In case of no days, $h_0$ should be returned.)

### Solution template

```python
def water_height(h0, r):
    # insert your code
    return h
```

| Input | |
|---|---|
| `h0` | A non-negative number with initial water height. |
| `r` | A vector with rain values (non-negative numbers) for a number of days. A vector may contain multiple, only one, or no elements. |

| Output | |
|---|---|
| `h` | The water height after the number of days has passed. |

### Example

Consider the input values $h_0 = 5$ and $\mathbf{r} = [\ 4.5 \quad 0 \quad 1.5 \quad 0 \quad 0 \quad 0.5 \quad 1 \quad 2 \quad 5\ ]$.

After the first from the number of days, the water height is $h_1 = 5 + 4.5 - 2 = 7.5$. After the second day we have $h_2 = 7.5 + 0 - 2 = 5.5$. After the third day $h_3 = 5.5 + 1.5 - 2 = 5$. After the fourth day $h_4 = 5 + 0 - 2 = 3$. After the fifth day $h_5 = 3 + 0 - 2 = 1$. After the sixth day $h_6 = 1 + 0.5 - 2 = -0.5$ which gets replaced by 0. After the seventh day $h_7 = 0 + 1 - 2 = -1$ which gets replaced by 0. After the eighth day $h_8 = 0 + 2 - 2 = 0$. After the ninth day $h_9 = 0 + 5 - 2 = 3$. This is the last day so the function should return

$$\underline{3}.$$

## Assignment 2 Astronomical season

If we ignore small variations, the four astronomical seasons start on the following days of the year: 20th of March (the first day of the astronomical spring), 21st of June (the first day of the astronomical summer), 23rd of September (the first day of the astronomical autumn), 21st of December (the first day of the astronomical winter).

### Problem definition

Write a function `astronomical_season` that takes a date in the format `dd/mm-yyyy` as input. The function should return the astronomical season corresponding to the given date. The return value should be either `spring`, `summer`, `autumn`, or `winter`.

### Solution template

```
def astronomical_season(date):
    # insert your code
    return season
```

| Input | |
|---|---|
| `date` | A string with the date in the format `dd/mm-yyyy`. |
| **Output** | |
| `season` | A string with the astronomical season, either `spring`, `summer`, `autumn` or `winter`. |

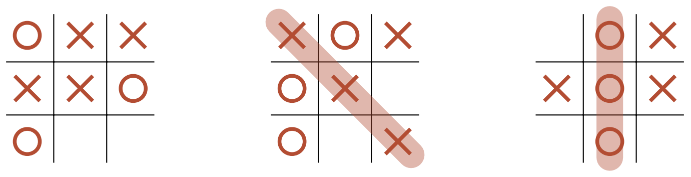
### Example

Consider an input `09/12-2020`, that is, the 9th of December 2020. This date occurs after the first day of astronomical autumn which starts with `23/09-2020`, but before (and not on) the first day of astronomical winter which starts with `21/12-2020`. Therefore, the function should return

`autumn`.

## Assignment 3 Tic-tac-toe

Tic-tac-toe (noughts and crosses) is a game for two players who take turns marking the fields in a $3 \times 3$ grid with either X or O. The winner is the player who succeeds in placing three marks in a row, which may be either horizontal, vertical, or diagonal, as shown in the illustration.



A tic-tac-toe board may be represented as a $3 \times 3$ matrix with numerical values 0 (an empty field), 1 (a field marked X) or 2 (a field marked O). Given such a matrix we want to know whether the board is valid, and for a valid board we want to know whether there is a winner. A board is valid if it could have been obtained by X placing the first mark, players taking turns, and then stopping the game at the first occurrence of a winning row.

In other words, the following conditions apply:

| | |
|---|---|
| X wins | There is exactly one winning row, and it is made by X. Furthermore, X placed the last mark, so there is exactly one more X mark than O marks. |
| O wins | There is exactly one winning row, and it is made by O. Furthermore, O placed the last mark, so there is equally many X an O marks. |
| No winner | There are no winning rows. Furthermore, there are either equally many X and O marks, or one more X mark than O marks. |
| Invalid board | All other situations. |

### Problem definition

Write a function `tictactoe` which takes a matrix representing a tic-tac-toe board as input. The function should return either -1 (invalid board), 0 (no winner), 1 (X wins) or 2 (O wins).

### Solution template

```
def tictactoe(board):
    # insert your code
    return score
```

**Input**

| | |
|---|---|
| `board` | A $3 \times 3$ matrix with numbers 0, 1 or 2. The matrix represents tic-tac-toe board. |

**Output**

| | |
|---|---|
| `score` | A number -1, 0, 1, or 2. The number indicates invalid board (-1), no winner (0) or the winner (1 or 2). |

### Example

Consider the matrix

$$\texttt{board} = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 1 & 2 \\ 2 & 0 & 0 \end{bmatrix}$$

which represents the leftmost tic-tac-toe board in the illustration above.

Neither X nor O has succeeded in placing three marks in row. So neither X or O wins.

We need to check whether the board is valid. The board contains 4 X-marks and 3 O-marks, so X has exactly one mark more than O marks. This means that the board is valid, as it could have been obtained by X making the first mark, and then players taking turns for three more marks each.

In conclusion, the function should return the value for *no winner* which is

$$\underline{0}.$$

## Assignment 4 Standardize address

You need to combine addresses from multiple sources, with different ways of writing the address line which includes the city name and the postal (zip) code. The differences are:

- Some write postal code first (like for example `2840 Holte`), while others write city first (for example `Holte 2840`).
- Some use a space to separate the parts of the address (like in the examples above), while others use underscore (like for example `2840_Holte`.)

You want to standardize the addresses such that the postal code always comes first, and that space is always used for separating the parts of the address. The city name may consist of multiple words, but you may assume that the city name only consists of letters and word separator (space or underscore). The postal code may be of different length, but you may assume that it only consists of digits. Furthermore, the postal code is always either at the beginning or at the end of the address line.

### Problem definition

Write a function `standardize_address` which takes as input an address line, which may lack standardization in terms of word separator and the ordering of address parts. The function should return the standardized address line.

### Solution template

```
def standardize_address(a):
    # insert your code
    return s
```

| | |
|---|---|
| **Input** | |
| `a` | A string with the address line which may lack standardization. |
| **Output** | |
| `s` | A string with the standardized address line. |

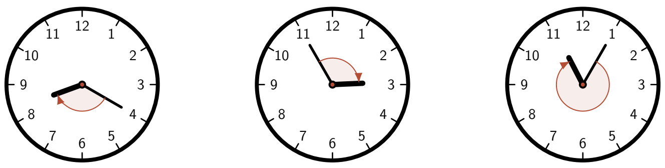
### Example

Consider the address line `New York 10001`. This address line uses spaces, so it requires no change in the word separator. However, the postal code `10001` is at the end of the address line after the city name `New York`. So the postal code needs to be moved to the beginning of the address line. The function should return the standardized address line

`10001 New York`.

## Assignment 5 Time angle

We want to keep track of when the two hands of the (analog) clock overlap. For this, we want to compute the angle from the minute hand to the hour hand for every minute of the day. The angle needs to be in a clockwise direction, meaning the angle which the minute hand should cover to reach the hour hand, as shown in the illustration.



Given a time of the day, the clockwise angles of the two hands (in degrees, and relative to the position pointing upwards) may be computed as

$$a_{\text{hour}} = \frac{1}{12}\left(h + \frac{m}{60}\right) \cdot 360^\circ \qquad \text{and} \qquad a_{\text{minute}} = \frac{m}{60} \cdot 360^\circ$$

where $h$ is the hour between 0 and 11, and $m$ is the minute between 0 and 59. If the hour is given by a number larger than 11, including the situation where the time is given in the 24-hour format, it needs to be reduced by subtracting 12 from its value. The angle from the minute hand to the hour hand may be computed as

$$a = a_{\text{hour}} - a_{\text{minute}}.$$

If this value is negative, the angle is counterclockwise. To get the clockwise angle, you need to add 360° to the computed value.

### Problem definition

Write a function `time_angle` that takes an hour and a minute for the time of day as input. The function should return a clockwise angle from the minute hand to the hour hand.

### Solution template

```python
def time_angle(hour, minute):
    # insert your code
    return angle
```

**Input**

| | |
|---|---|
| `hour` | An integer between 0 and 23 representing an hour for the time of day. |
| `minute` | An integer between 0 and 59 representing a minute for the time of day. |

**Output**

| | |
|---|---|
| `angle` | An angle (in degrees) from the minute hand to the hour hand (in clockwise direction). |

### Example

Consider the input `hour` = 8, `minute` = 20 (as on the leftmost clock in the illustration above). The hour value is not larger than 11, so we do not need to reduce it. We compute

$$a_{\text{hour}} = \frac{1}{12}\left(8 + \frac{20}{60}\right) \cdot 360^\circ = 250^\circ \qquad \text{and} \qquad a_{\text{minute}} = \frac{20}{60} \cdot 360^\circ = 120^\circ,$$

$$a = a_{\text{hour}} - a_{\text{minute}} = 250^\circ - 120^\circ = 130^\circ.$$

The computed value is not negative, so we do not need to convert it to a clockwise angle. Therefore, the function should return

$$\underline{130}.$$