Technical University of Denmark

Course name	Introduction to programming and data processing
Course number	02631 (02632, 02633, 02634, 02692)
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 CodeJudge

   `https://dtu.codejudge.net/prog-e20/assignments`

   under *Exam*. When you hand in 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 DTU's Online exam site. Each assignment must be uploaded as one separate .py file, given the same name as the function in the assignment:

   (a) `robust_mean.py`

   (b) `shape_roundness.py`

   (c) `candy_exchange.py`

   (d) `change_case.py`

   (e) `nearest_shop.py`

   The files must be handed in separately (*not* as a zip-file) and must have these exact filenames.

After the exam, your solutions submitted to DTU Inside 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 Robust mean

The mean of the values x_n where $n = 1, \ldots, N$ may be affected by outliers. To ignore outliers, you may compute a robust mean using the following procedure. First, compute the mean value μ and the standard deviation σ as

$$\mu = \frac{1}{N}\sum_{n=1}^{N} x_n\,, \qquad \sigma = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(x_n - \mu)^2}\,.$$

Then consider only values x_n which deviate not more than σ from μ and discard the other values (the outliers). That is, you should keep the values where

$$\mu - \sigma \leq x_n \leq \mu + \sigma.$$

Let's say that the values you keep are x_k, $k = 1, \ldots, K$. The robust mean r is now computed as the mean of these values

$$r = \frac{1}{K}\sum_{k=1}^{K} x_k\,.$$

Problem definition

Create a function `robust_mean` which takes a vector with some values as input, and returns a robust mean of the values.

Solution template

```python
def robust_mean(x):
    # insert your code
    return r
```

Input	
x	A vector containing values x_n.
Output	
r	A scalar with the robust mean of x_n.

Example

Consider the values

$$\mathbf{x} = [\,5.4\,,\quad 17.4\,,\quad 5.5\,,\quad 6.4\,,\quad 4.3\,]\,.$$

The mean and the standard deviation are

$$\mu = \frac{1}{5}(5.4 + 17.4 + 5.5 + 6.4 + 4.3) = 7.8$$

$$\sigma = \sqrt{\frac{1}{5}\left((5.4 - 7.8)^2 + (17.4 - 7.8)^2 + (5.5 - 7.8)^2 + (6.4 - 7.8)^2 + (4.3 - 7.8)^2\right)} = 4.846\,.$$

We have $\mu - \sigma = 2.9540$ and $\mu + \sigma = 12.6460$. Only one value, the value 17.4, is to be discarded as outlier as it is too big. The robust mean is

$$r = \frac{1}{4}(5.4 + 5.5 + 6.4 + 4.3) = \underline{5.4}\,.$$

Assignment 2 Shape roundness

A shape may be represented using a Boolean (logical) matrix **S**, where `True` (`1`) represents the shape and `False` (`0`) represents the background. You can imagine this being a pattern of quadratic black-and-white tiles or image pixels.

One measure of the shape roundness is the ratio between the shape area A and the shape perimeter P

$$r = \frac{A}{P}.$$

Here, the area A is the sum of areas of all the squares of the shape, while the perimeter P is the sum of lengths of all the edges delineating the shape.

Problem definition

Create a function `shape_roundness` which takes a Boolean matrix representing the shape as input, and returns the measure of the shape roundness.

Solution template

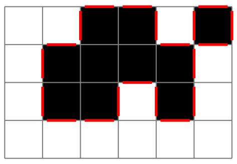
```python
def shape_roundness(S):
    # insert your code
    return r
```

Input	
`S`	A Boolean matrix (elements are either `True` (`1`) of `False` (`0`)) representing the shape.
Output	
`r`	A decimal number with the shape roundness.

Example

Consider the matrix **S** (we write 0 for `False` and 1 for `True`) and the shape it represents.

$$\mathbf{S} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



The area of the shape is obtained by counting the black squares. For the matrix **S** the area is $A = 10$. The perimeter of the shape is obtained by counting the vertical and horizontal red edges delineating the shape. For the matrix **S** the perimeter is $P = 20$.

Shape roundness is

$$r = \frac{10}{20} = \underline{0.5}.$$

Assignment 3 Candy exchange

The owner of a candy shop came up with a plan for reducing the littering in front of the shop: "Bring back five pieces of candy wrapping, and get one candy for free!" This poses a question: If you have a certain number of candies, how many candies can you actually end up eating? Here we assume that you keep returning the wrappers as long as possible.

Problem definition

Create a function `candy_exchange` which takes the number of candies you have as an input, and returns the number of candies you can end up eating.

Solution template

```
def candy_exchange(n):
    # insert your code
    return m
```

Input	
`n`	An integer with the number of candies you have.

Output	
`m`	An integer with the number of candies you can end up eating.

Example

Let's say you start with $n = 73$ candies.

After eating these candies you are left with 73 wrappers. Your status is: 73 eaten, and 73 wrappers.

You exchange 70 wrappers for 14 new candies which you eat. Your status is: 87 eaten (73 from before and 14 new), and 17 wrappers (3 which you couldn't exchange and 14 new).

You exchange 15 wrappers for 3 new candies which you eat. Your status is: 90 eaten (87 from before and 3 new), and 5 wrappers (2 which you couldn't exchange and 3 new).

You exchange 5 wrappers for one new candy which you eat. Your status is 91 eaten (90 from before and 1 new), and 1 wrapper, which you cannot exchange for more candies.

The status is summarized below.

candies eaten	73	87	90	91
wrappers	73	17	5	1

You can end up eating

$$m = \underline{91}$$

candies.

Assignment 4 Change case

Two naming conventions for multi-word variable names are: snake case and camel case. Snake case uses underscores in the place of a space. Camel case indicates the separation of the words using a single capitalized letter. An example of snake case is `circle_radius`, and the corresponding camel case is `circleRadius`. Single-word variable names are the same in the two conventions.

Problem definition

Create a function `change_case` which takes a string with the variable name in either a snake case or a camel case, and returns a string with the variable name in the other case. The variable name may be single-word or multi-word.

Solution template

```
def change_case(v):
    # insert your code
    return u
```

Input	
v	A string with the variable name in either a snake case or a camel case. The variable name may be single-word or multi-word.
Output	
u	A string with the variable name in the other naming convention than that used in the input string.

Example

Consider the variable name

$$v = \text{'thisIsIdentifierName'}.$$

The presence of capital letters reveals that this is a camel case. The full name can be divided into individual words

this is identifier name

Written using snake case the variable name is

$$u = \underline{\text{'this\_is\_identifier\_name'}}.$$

Assignment 5 Nearest shop

A chain of shops uses post code to direct customers to the nearest shop. For this, they have a list with 4-digit post codes of their shops. When the customer enters a 4-digit post code, the post code of the nearest shop is found like this:

- If the customer's post code exactly matches one of the shop post codes, this is the nearest shop.
- If no exact match is found, consider shops matching the customer's post code in the first three digits. If there are 3-digit matches, the match appearing first in the list is the nearest shop.
- If no 3-digit matches are found, consider 2-digit matches in a similar manner. If no 2-digit matches are found, consider 1-digit matches.
- If no shop is matching the first digit of the customer's post code, the shop appearing first in the list is the nearest shop.

Problem definition

Create a function `nearest_shop` that takes one 4-digit integer and a vector containing 4-digit integers. The function should return the element from the vector which identifies the nearest shop.

Solution template

```python
def nearest_shop(c, s):
    # insert your code
    return n
```

Input	
`c`	A 4-digit integer representing the customer's post code.
`s`	A vector containing 4-digit integers representing post codes of the shops.
Output	
`n`	A 4-digit integer representing the post code of the nearest shop.

Example

Consider the customer's post code `c` = 3490 and a list of shop post codes

$$\texttt{s} = [2300, \quad 1890, \quad 3990, \quad 3590, \quad 7900].$$

In the list with shop post codes, there is no exact match for 3490. There are no shop post codes starting with 349 or with 34. There are two post codes starting with 3: the number 3990 and the number 3590. Of those two numbers, the one appearing first in the list should be returned and that is the number

$$\texttt{n} = \underline{3990}.$$