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-jan20/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) `riemann.py`
   (b) `packing.py`
   (c) `dice_sum_count.py`
   (d) `imputed_sum.py`
   (e) `tokenize.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 A Riemann series

Consider the convergent series

$$s_a = \sum_{i=1}^{n} \frac{(-1)^{i+1}}{\lceil i/2 \rceil} = 1 - 1 + 1/2 - 1/2 + 1/3 - 1/3 + \dots \tag{1}$$

$$s_b = \sum_{i=1}^{n} \frac{(-1)^{i+1}}{i} = 1 - 1/2 + 1/3 - 1/4 + 1/5 + \dots \tag{2}$$

where $\lceil\ \rceil$ is the ceil function. We want to compute the sum after n terms from one of the series.

Problem definition

Create a function called `riemann` which takes as input: n that is the number of terms and the form specified as either "a" or "b" corresponding to equations 1 or 2, respectively. The function should return the sum of the series from the n number of terms, depending on the form.

Solution template

```
def riemann(n, form):
  #insert your code
  return s
```

Input	
`n`	Scalar with the number of terms (integer, larger than zero)
`form`	String that is either "a" or "b".

Output	
`s`	Scalar with the sum of the series (float)

Example

Consider the example with the number of terms $n = 7$ and the form = "a"

$$s_a = \sum_{i=1}^{7} \frac{(-1)^{i+1}}{\lceil i/2 \rceil} = 1 - 1 + 1/2 - 1/2 + 1/3 - 1/3 + 1/4 \tag{3}$$

$$= 0 + 0 + 0 + 0.25 = 0.25. \tag{4}$$

The function returns

$$\mathtt{s} = 0.25 \tag{5}$$

A

Assignment B Packing

We want to test if a two-dimensional box-shaped object can fit in a two-dimensional box. The sizes of the box and object are each specified with a two-dimensional vector. We want to know the amount of missing space for each dimension, which is the difference between the object size and the box size. If there is enough space in a dimension the function should return 0 for that dimension. The object can be rotated and only the case with the best fit should be returned.

Problem definition

Create a function called `packing` which takes as input: A 2-dimensional vector with the size of the box and a 2-dimensional vector with the size of the object. The function should return the missing fit computed as the difference in each dimension. If there is enough space in a dimension then that dimension should be set to zero. The object can be rotated and only the case with the smallest sum should be returned. The sum is computed as the sum over the elements of the missing fits. If the two sums are equal, then the non-rotated version is to be preferred.

Solution template

```
def packing(box, object):
  # insert your code
  return d
```

Input	
`box`	2-dimensional vector with size specification of the box.
`object`	2-dimensional vector with size specification of the object.

Output	
`d`	2-dimensional vector with missing fit.

Example

Regard the box, $\mathbf{b} = [2, 5]$, and the object, $\mathbf{g} = [4, 2.1]$. When rotated, the size of the object becomes $\tilde{\mathbf{g}} = [2.1, 4]$. The missing space is computed as $\mathbf{d}$ (for the non-rotated case) and $\tilde{\mathbf{d}}$ (for the rotated case):

$$\mathbf{d} = \mathbf{g} - \mathbf{b} = [4, 2.1] - [2, 5] = [2, -2.9] \tag{6}$$

$$\tilde{\mathbf{d}} = \tilde{\mathbf{g}} - \mathbf{b} = [2.1, 4] - [2, 5] = [0.1, -1] \tag{7}$$

The dimensions where there is enough space are set to zero:

$$\mathbf{d}' = [2, 0] \tag{8}$$

$$\tilde{\mathbf{d}}' = [0.1, 0] \tag{9}$$

The sums of the element of these two vectors are $2 + 0 = 2$ for the non-rotated case and $0.1 + 0 = 0.1$ for the rotated case, i.e., the rotated case is the one that fits best and the associated missing fit is returned:

$$\mathtt{d} = [0.1, 0] \tag{10}$$

Assignment C Dice sum count

For a game of dice with one, two or three dice, we want to count how many times that the sum of the dice is equal or below a certain threshold when counting all the possible combination of the dice.

Problem definition

Create a function called `dice_sum_count` that takes two inputs: The threshold and the number of dice. The function should return the number of dice sum combinations that are equal or below a given threshold.

Solution template

```
def dice_sum_count(threshold, dice):
  # insert your code
  return count
```

Input	
`threshold`	Scalar with the threshold (float)
`dice`	Scalar with the number of dice, either 1, 2 or 3 (integer)

Output	
`count`	Scalar with the number of sums equal or below the threshold (integer)

Example

Consider the case where the threshold is 4 and number of dice are 2. The 36 possible combination of dice are:

$$[1,1],[1,2],[1,3],[1,4],[1,5],[1,6],[2,1],[2,2],[2,3],\ldots,[3,1],[3,2],\ldots,[6,5],[6,6] \tag{11}$$

The sum of each of these games are

$$2,3,4,5,6,7,3,4,5,\ldots 4,5,\ldots 11,12 \tag{12}$$

The sums that are below or equal to 4 is:

$$2,3,4,3,4,4, \tag{13}$$

There are 6 sums below or equal to 4 and the number 6 is returned as

$$\texttt{count} = 6 \tag{14}$$

C

Assignment D Imputated sum

For a series of numbers, we want to compute the sum. A problem with the series is that some of the numbers are "missing" indicated with the number 999. The "missing" numbers need to be computed as the mean of the immediate neighboring values.

Consider an example series of numbers $\mathbf{x}$,

$$\mathbf{x} = [17, -4.1, 3, 999, 7, 18, 999] \tag{15}$$

Here the 4th and the last element are "missing". The 4th element is computed as the mean of the neighboring values 3 and 7, i.e., 5, and the last element is set as the neighboring values 18. This results in a new imputed series:

$$\tilde{\mathbf{x}} = [17, -4.1, 3, 5, 7, 18, 18] \tag{16}$$

The sum of these numbers is 63.9.

Problem definition

Create a function `imputed_sum` which takes a vector with numbers and where missing values are indicated with the value 999. The missing values must be computed as the mean of the two immediate neighboring values. If the missing value is in the very start of the series or the very end, then the missing value should be set to the value of its immediate neighbor. The series may contain zero or more missing values, but it can be assumed that no two missing values follow each other immediately and that there is always one non-missing value.

Solution template

```
def imputed_sum(x):
    #insert your code
    return s
```

Input	
`x`	Vector with series of numbers, where missing values are indicated with 999.

Output	
`s`	Scalar with sum.

Example

For the example above we have

$$\mathbf{x} = [17, -4.1, 3, 999, 7, 18, 999] \tag{17}$$

The resulting sum should be returned as

$$\mathbf{s} = 63.9 \tag{18}$$

D

Assignment E Tokenization

Given a string representing a text from a Breton-like language, we want to find the count of individual words. We want to handle the case with an apostroph (a single quotation mark) between "c" followed by "h", i.e., "c'h". At the point with the apostroph between these two letters the word should not be split. For other letter combinations with apostroph, the string should be split into separate words.

Problem definition

Create a function called `tokenize` which takes a string as input. The output of the function should return the number of words in the string. The word separation is where there is a space or an apostroph, except when the apostroph is between "c" followed by "h". It can be assumed that all characters are either the letters from a to z, the space character or the apostroph. It can also be assumed that there is never two spaces or two apostrophes right after each other, that the apostroph is always between letters and that there is always one or more words.

Solution template

```
def tokenize(text):
  # insert your code
  return words
```

Input

`text`	String representing a text.

Output

`words`	Skalar with the number of words in the text (integer).

Example

Consider the string

"`ma n'oc'h ket asur eus an lec'h`"

For this example, we split the string into the words

`"ma", "n", "oc'h", "ket", "asur", "eus", "an", "lec'h"`

Here there are 8 words and this is returned as

$$\texttt{words} = 8 \tag{19}$$

E