Technical University of Denmark

Course name	Introduction to programming and data processing
Course number	02633 (02631, 02632, 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-jun21/assignments/

   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 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) `node_divergence.py`

   (b) `blood_pressure.py`

   (c) `closest_match.py`

   (d) `which_tetrahedral.py`

   (e) `flowering_plants.py`

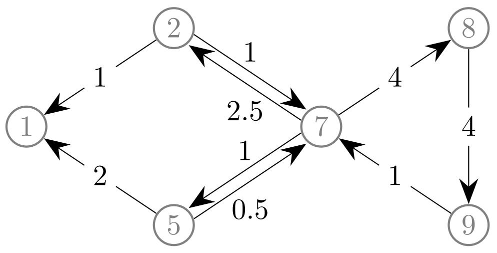
   The files must be handed in separately (*not* as a zip-file) and must have these exact filenames.

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 Node divergence

A graph can be represented using a matrix where every row contains a triplet of numbers (i, j, w_{ij}) representing one graph edge. Here, i is an index of from-node, j is an index of to-node, and w_{ij} is the weight of the edge from i to j. For example, consider the graph in the illustration and its representation using matrix A.



$$A = \begin{bmatrix} 2 & 7 & 1 \\ 2 & 1 & 1 \\ 5 & 1 & 2 \\ 5 & 7 & 0.5 \\ 7 & 2 & 2.5 \\ 7 & 5 & 1 \\ 7 & 8 & 4 \\ 9 & 7 & 1 \\ 8 & 9 & 4 \end{bmatrix}$$

Divergence of node i is defined as

$$d_i = \sum_{\substack{j \\ \text{edge } ij}} w_{ij} - \sum_{\substack{j \\ \text{edge } ji}} w_{ji}.$$

So d_i is the difference between the sum of weights of all edges originating from i and the sum of weights of all edges ending in i. For example, the divergence of node 7 in the above graph is $d_7 = (2.5+1+4)-(1+0.5+1) = 5$.

Problem definition

Create a function `node_divergence` which takes a matrix representing a graph as input. The function should return a matrix containing sorted indices for graph nodes in one column and the divergence values for the corresponding nodes in the second column.

Solution template

```python
def node_divergence(A):
    # insert your code
    return D
```

Input

`A`	$N \times 3$ matrix representing a graph, where N is the number of edges. Node indices are integers, but do not need to form an interval.

Output

`D`	$M \times 2$ matrix with nodes and their divergences, where M is the number of nodes. Every row of this matrix contains a pair of numbers (i, d_i) where i is a node index, and d_i is a divergence of node i. The rows should be sorted according to the node indices.

Example

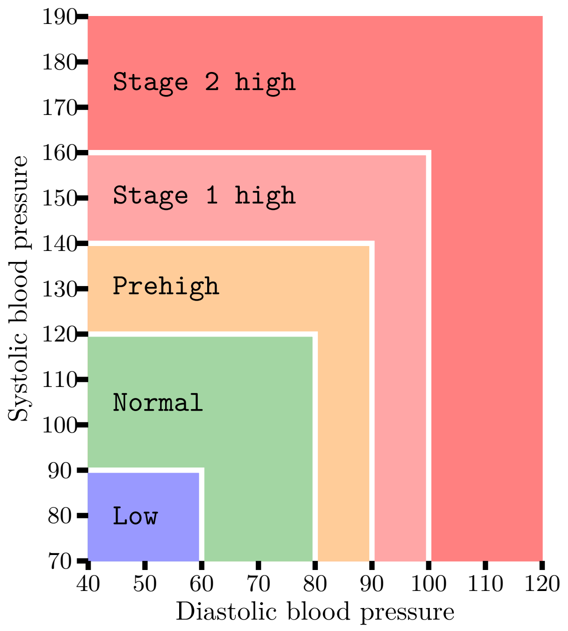
Consider matrix A from the illustration as input. By finding unique integers from the first two columns of A, we identify 6 node indices: 1, 2, 5, 7, 8, 9.

Node with index 1 has two edges, represented by the second and third row of A. Both of these edges are ending in node 1, and their weights are 1 and 2, so we compute $d_1 = -(1+2) = -3$. Node with index 2 has two out-going edges (in first and second row) and one in-going edge (in fifth row) so $d_2 = (1+1)-2.5 = -0.5$. A similar computation is performed for the remaining graph nodes and the function should return a matrix

$$D = \begin{bmatrix} 1 & -3 \\ 2 & -0.5 \\ 5 & 1.5 \\ 7 & 5 \\ 8 & 0 \\ 9 & -3 \end{bmatrix}$$

Assignment 2 Blood pressure

Measurement of blood pressure consists of two values: systolic blood pressure (a higher value) and diastolic blood pressure (a lower value). Based on these two values, blood pressure can be categorized as indicated in the chart below.



Problem definition

Create a function `blood_pressure` which takes a number s with systolic blood pressure and a number d with diastolic blood pressure as input. The function should return a string with blood pressure category, written exactly as in the chart above. If a measure is between categories, a category for a higher value should be returned.

Solution template

```
def blood_pressure(systolic, diastolic):
    # insert your code
    return category
```

Input	
`systolic`	A number with a valid systolic blood pressure.
`diastolic`	A number with a valid diastolic blood pressure.

Output	
`category`	A string with the blood pressure category.

Example

If the systolic blood pressure is 120 and the diastolic blood pressure is 65, according to the chart the measurement is between the normal blood pressure and the prehigh blood pressure. The function should return the higher category, which is the string

```
Prehigh
```

Assignment 3 Closest match

Given a query word, and a collection of words, we want to find a word from the collection, which is closest to the query word.

To compute the distance between the query word and any of the candidate words from the collection, we use two measures: the number of shared letters and the number of excess letters. Shared letters are counted from the beginning of the words (query and candidate), stopping at the first difference. Excess letters are letters from the longer of the words (either query or candidate) which are not shared. For example, if `siesta` is query, and `sinus` is candidate, the distance is 2 shared letters (`si`) and 4 excess letters (`esta`).

Rules for finding the word which is closest to the query are explained by considering the query and two words from the collection. Rule 1: The word from the collection with the larger amount of shared letters is closer to the query. Rule 2: If two words from the collection have the same amount of shared letters, the word with the smallest amount of excess letters is closer to the query. Rule 3: If two words from collection have the same amount of shared letters and the same amount of excess letters, the word which appears first in the collection is closer to the query.

Problem definition

Create a function `closest_match` which takes a string with a query word, and a string with a collection of words as input. The function should return the word from the collection which is the closest match to the query.

Solution template

```python
def closest_match(query, collection):
    # insert your code
    return match
```

Input	
`query`	String with the query word containing only lower case characters.
`collection`	String with a collection of words separated by a single space character. Words contain only lower case characters.
Output	
`match`	String with the word from collection which is the closest match to query.

Example

Consider query `aboriginal` and a collection

`digestive abracadabra abudhabi acapulco popcorn`

Counting the number of shared letters between the query and every word in the collection we get the sequence

`0, 2, 2, 1, 0`

The candidates for the closest match are therefore words `abracadabra` and `abudhabi` which share 2 letters (letters `ab`) with the query. The number of excess letters for `abracadabra` is 9 (letters `racadabra`) and for `abudhabi` is 8 (letters `original`). So word `abudhabi` with 2 shared and 8 excess letters is the closest match to the query. The function should return

`abudhabi`

Assignment 4 Which tetrahedral

A n-th tetrahedral number is given by

$$T_n = \frac{n(n+1)(n+2)}{6}$$

and a sequence of tetrahedral numbers is

$$1,\ 4,\ 10,\ 20,\ 35,\ 56,\ 84,\ 120,\ 165,\ 220\ldots$$

Problem definition

Create a function `which_tetrahedral` that given a tetrahedral number returns its index (its position in the sequence). If the input is not a tetrahedral number, the function should return 0.

Solution template

```python
def which_tetrahedral(t):
     # insert your code
     return n
```

Input	
`t`	Non-negativ number which is presumed to be tetrahedral.
Output	
`n`	Index of the tetrahedral number, or 0 if the number is not tetrahedral.

Example

Consider number $t = 121$. Since t is strictly between the tetrahedral number 120 (with index 8) and 165 (with index 9), we can conclude that t is not tetrahedral so function should return

$$\underline{0}$$

Assignment 5 Flowering plants

The information about the flowering season for the plants in a tropical greenhouse is stored in a $N \times 2$ matrix of integer numbers between 0 and 12. Each row in this matrix is a pair of numbers representing one plant. The first number is a month of the year where the plant starts flowering and the second number is a month where the plant stops flowering. For example a pair $(11, 2)$ represent a plant with flowering season starting in November (month 11) and ending in February (month 2). If a plant flowers all year round, it is represented with $(0, 0)$. A plant represented with $(7, 7)$ has a short flowering season which both starts and ends in July.

Problem definition

Create a function `flowering_plants` which takes as input a matrix of greenhouse plants G, and a month m given by a number between 1 and 12. The function should return the total number of plants which flower in month m. Plants which start or stop flowering in the month m should be counted as flowering in month m.

Solution template

```
def flowering_plants(G,m):
    # insert your code
    return n
```

Input	
G	$N \times 2$ matrix of numbers between 0 and 12 representing a flowering season of plants.
m	Number between 1 and 12 representing a month.
Output	
n	Number of plants flowering in the given month.

Example

Consider the input

$$G = \begin{bmatrix} 5 & 9 \\ 9 & 11 \\ 12 & 6 \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad m = 6$$

The first plant has flowering season represented by $(5, 9)$ which corresponds to months $5, 6, 7, 8, 9$, so this plant flowers in June. The second plant is represented by $(9, 11)$ which corresponds to $9, 10, 11$, so it does not flower in June. The third plant is $(12, 6)$ corresponding to $12, 1, 2, 3, 4, 5, 6$, so it flowers in June. The last plant is $(0, 0)$, and it flowers all year round, also in June. In total there are 3 plants flowering in June. The function should return $\underline{3}$.