Technical University of Denmark

Course name	Introduction to programming and data processing Programming and data processing (second programming language)
Course number	02632, 02633, 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 upload your solutions individually on CodeJudge (`dtu.codejudge.net/prog-aug15/assignment`) under *Afleveringer/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 your solutions on CampusNet. Each assignment must be uploaded as one separate .py file, given the same name as the function in the assignment:

   (a) `classifyBMI.py`
   (b) `cumulativeStock.py`
   (c) `RPNCalculator.py`
   (d) `starPoints.py`
   (e) `matrixCleanup.py`

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

After the exam, your solutions will be automatically evaluated on CodeJudge on a range of different tests, to check that they work correctly in general. The assessment of you 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.
- 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 Body mass index

Body mass index (BMI) is commonly used to classify underweight, overweight and obesity in adults. The body mass index b is defined as the weight w (in kilograms) divided by the squared height h (in metres),

$$b = \frac{w}{h^2} \tag{1}$$

The following table lists the names of different BMI groups:

BMI group	BMI, b
`Severely underweight`	$b < 16$
`Underweight`	$16 \leq b < 18.5$
`Normal`	$18.5 \leq x < 25$
`Overweight`	$25 \leq x < 30$
`Obese`	$30 \leq x < 40$
`Severely obese`	$40 \leq x$

Problem definition

Create a function named `classifyBMI` that takes the height and weight as numeric inputs and returns the BMI group as a string (written exactly as in the table above.)

Solution template

```
def classifyBMI(height, weight):
  #insert your code
  return BMIGroup
```

Input

`height`	Height in meters (decimal number).
`weight`	Weight in kilograms (decimal number).

Output

`BMIGroup`	BMI group (string).

Example

If the height is $h = 1.85$ meter and the weight is $w = 88$ kilograms, the BMI can be computed as $b = \frac{88}{1.85^2} = 25.71$ and the string `Overweight` must be returned.

A

Assignment B Stock management

You are working with a stock management system that tracks how many units of some item a store has in stock. As input you are given a liste of transactions, i.e. a list of numbers (integers): Positive numbers indicate that items are added to the stock, and negative numbers indicate that items are removed from the stock. The number zero indicates that the stock is reset to zero. Before the first transaction, the number of items in stock is zero.

Problem definition

Create a function named `cumulativeStock` which returns the number of items in stock after each transaction given a vector containing the list of transactions.

Solution template

```
def cumulativeStock(transactions):
  #insert your code
  return stock
```

Input	
`transactions`	Transactions (vector of whole numbers).
Output	
`stock`	Number of units in stock after each transition (vector of whole numbers).

Example

Consider the following list of transactions:

$$10, -4, -3, 10, -12, 0, 8$$

Initially the stock is 0 so after the first transaction there are 10 items in stock. At the second transaction 4 items are removed, so the stock is down to 6 items. Next, 3 items are removed, so we are down to 3 items in stock. Then 10 items are added and we are up to 13 items, whereafter 12 items are removed and we are down to 1 item. At the next transaction the stock is reset to 0, and finally 8 items are added. The total stock after each transaction is thus:

$$10, 6, 3, 13, 1, 0, 8$$

B

Assignment C Reverse Polish notation calculator

Reverse Polish notation is way to write mathematical expressions in which the operators (such as plus and minus etc.) follow after the operands (the numbers they operate on). For example, adding the numbers 3 and 4, would be expressed as "`3 4 +`".

In this assignment you will make a simple reverse Polish calculator that works in the following way. The calculator has a list of numbers (called a stack) which initially is empty. If the calculator is given a number, it adds it to the end of the stack. If the calculator is given an operation to perform, it extracts the two last numbers from the stack, performs the operation on those two numbers, and adds the result to the end of the stack. The calculator must support the following operations:

	Operation
`+`	Addition
`-`	Subtraction
`s`	Swap

Problem definition

Create a function named `RPNCalculator` that takes as input a text string defining an sequence of commands in reverse Polish notation. The commands are either numbers or one of the strings `+`, `-`, `s`, and are separated by space. The function must output the final state of the stack (as a vector) after performing the given operations.

Solution template

```
def RPNCalculator(commands):
  #insert your code
  return stack
```

Input	
`commands`	Sequence of commands (string).
Output	
`stack`	Value of the stack (vector).

Example

Consider for example the following sequence of commands as input:

```
2 10 17 - s
```

The first command puts the number 2 on the stack. Then the number 10 is put on the stack, and next the number 17 is put on the stack. The stack now contains the numbers 2, 10, and 17. Next, the command `-` (subtraction) operates on the two last numbers (10 and 17), removes these numbers from the stack, subtracts 17 from 10 yielding the result -7, and puts this result on the stack, which now contains the numbers 2 and -7. Finally, the command `s` (swap) instructs the calculator to swap the two last numbers on the stack, which then contains the numbers -7 and 2.

command:	2	10	17	-	s
stack:	2	2	2	2	-7
		10	10	-7	2
			17		

The final result is thus the vector:

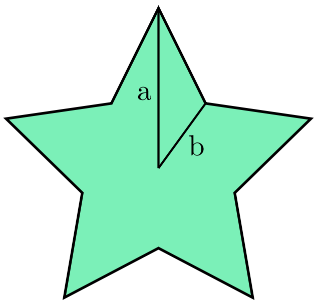
$$[-7,\ 2]$$

Assignment D Star area

The area of an n-pointed star is given by

$$A = n \cdot a \cdot b \cdot \sin\left(\frac{\pi}{n}\right) \tag{2}$$

where a and b are the distances from the center to the outer and inner vertices of the star respectively, and n is the number of points of the star. Here, a 5-pointed star is illustrated:



Problem definition

Create a function named `starPoints` that computes the maximum number of points for a star with the given values of a and b such that the area of the star does not exceed the specified maximum area.

Solution template

```python
def starPoints(a, b, maxArea):
  #insert your code
  return points
```

Input	
`a`	Distance from center to outer vertices (positive decimal number)
`b`	Distance from center to inner vertices (positive decimal number)
`maxArea`	The maximum area of the star (positive decimal number)
Output	
`points`	Number of points on the star (integer)

Example

Consider $a = 2$, $b = 1$ and a maximum area of 6.1. For different values of n the area can be computed as shown below:

n	3	4	5	6	7	8	9	10
A	5.196	5.657	5.878	6.000	6.074	6.123	6.156	6.180

The largest number of points n for which the area does not exceed 6.1 is $n = 7$. Thus, the function should return the value 7.

D

Assignment E Removing rows and columns

You are working with a data set in the form of a rectangular array of positive numbers (a matrix M). Some of the values in the array are unknown, and these have been marked by the value zero.

Problem definition

Create a function named `matrixCleanup` which takes a matrix as input and returns the matrix where the rows and columns that contain one or more zero values have been removed.

Solution template

```
def matrixCleanup(M):
  #insert your code
  return MClean
```

Input	
`M`	Input matrix (numpy array)

Output	
`MClean`	Output matrix (numpy array)

Example

Consider the following input matrix:

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 0 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 0 & 0 & 19 & 20 \end{bmatrix}$$

Since row 2 and 4 contain one or more zeros these rows must be removed, and since column 2 and 3 contain one or more zeros these columns must also be removed.

$$M = \begin{bmatrix} 1 & \cancel{2} & \cancel{3} & 4 & 5 \\ \cancel{6} & \cancel{0} & \cancel{8} & \cancel{9} & \cancel{10} \\ 11 & \cancel{12} & \cancel{13} & 14 & 15 \\ \cancel{16} & \cancel{0} & \cancel{0} & \cancel{19} & \cancel{20} \end{bmatrix}$$

Thus we are left with the following, which should be the output of the function:

$$M_{\text{Clean}} = \begin{bmatrix} 1 & 4 & 5 \\ 11 & 14 & 15 \end{bmatrix}$$