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 upload your solutions individually on CodeJudge (`dtu.codejudge.net/prog-aug18/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) `nextTriangular.py`

   (b) `tictactoe.py`

   (c) `blackjack.py`

   (d) `hypergeo.py`

   (e) `formatName.py`

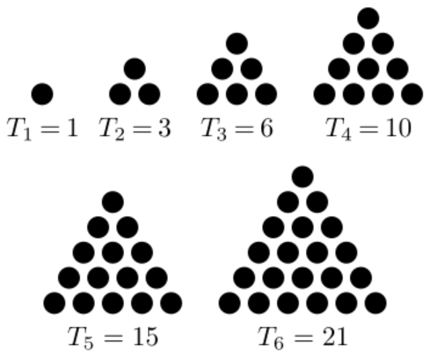
   The files must be handed in separately (*not* as a zip-file) and must have these exact filenames.

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 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.
- 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 Triangular numbers

An *triangular number* corresponds to the number of objects that can be arranged in a triangular shape, as shown below.



The first 6 triangular numbers are 1, 3, 6, 10, 15, 21. All triangular numbers can be computed using the formula:

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

Problem definition

Create a function named `nextTriangular` that takes as input a number x, and returns the smallest triangular number greater than or equal to x.

Solution template

```
def nextTriangular(x):
  #insert your code
  return triangular
```

Input	
`x`	Start number, (integer.)
Output	
`triangular`	Smallest triangular number greater than or equal to x (positive integer.)

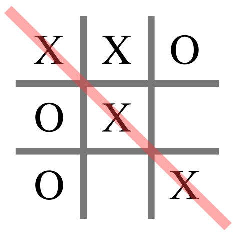
Example

Consider the input number $x = 17$. The smallest triangular number that is greater than or equal to x is <u>21</u> which the function should return.

A

Assignment B Tic-tac-toe

Tic-tac-toe is a game for two playes, X and O. The board has 3-by-3 fields, and the players take turn marking a field. The player who first gets three marks in a horizontal, vertical or diagonal row wins the game.



In this assignment we represent the board by a 3-by-3 matrix. A field can be empty or marked by X or O, which we represent by 0, 1, and -1 respectively. For example, the board in the figure is represented by the following matrix:

$$\begin{bmatrix} 1 & 1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

Problem definition

Create a function named `tictactoe` that takes as input a 3×3 matrix that represents a board. If either of the players has three marks in a row, the function must set all other fields on the board to zero, and return the board which now contains only the three "winning" marks. If no player has three marks in a row, the function must return the original board. You may assume that there is no more than one winning row on the board.

Solution template

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

Input	
`board`	Board (3×3 matrix containing values 0, 1, and -1.)
Output	
`newBoard`	Updated board (3×3 matrix containing values 0, 1, and -1.)

Example

Consider the input board shown in the figure above. Since X has a winning diagonal row, all other fields on the board must be set to zero, and the function must return

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Assignment C Blackjack

Blackjack is a card game which can be played against a dealer at a casino. The player is dealt a number of cards, and the objective is to reach a higher score than the dealer, without exceeding a score of 21. The score for the cards 2–10 is their value (2–10), whereas face cards have a score of 10 points. An ace is worth 1 point or 11 points, whichever leads to the higher score without exceeding 21. The total score of a player is the sum of the scores for each card in his hand.

Problem definition

Create a function named `blackjack` that takes as input an array of numbers representing the cards held by the player. The cards are represented in the array by numbers 1–10 corresponding to their point value. An ace is represented by 1. The function must return the total score (less than or equal to 21). If the score exceeds 21, the function must return 0.

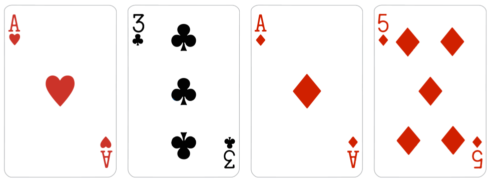
Solution template

```
def blackjack(hand):
  #insert your code
  return score
```

Input	
`hand`	Hand of cards (array with numbers 1–10.)
Output	
`score`	Score of the hand (whole number 0–21.)

Example

Consider the following hand, which is represented as 1,3,1,5.



If both aces count as 1, the total score would be $1 + 3 + 1 + 5 = 10$. If one of the aces count as 11, the total would be $11 + 3 + 1 + 5 = 20$. If both aces count as 11, the total would be $11 + 3 + 11 + 5 = 30$. Since the score 20 does not exceed 21, the total (best) score of this hand is <u>20</u> which the function should return.

C

Assignment D Hypergeometric function approximation

The socalled hypergeometric function can be defined by the following infinite series

$$F(a, b; c; z) = \sum_{n=0}^{\infty} \frac{(a)_n (b)_n}{(c)_n} \cdot \frac{z^n}{n!},$$

where the parameters a, b and c are numbers which can take any value. The series is defined for arguments $|z| < 1$. Here $n!$ denotes the factorial function, and the notation $(a)_n$ denotes the rising Pochhammer symbol, defined as

$$(a)_n = \begin{cases} 1, & n = 0 \\ a(a+1)\cdots(a+n-1), & n = 1, 2, 3, \ldots \end{cases}$$

or equivalently defined as

$$(a)_n = \frac{\Gamma(a+n)}{\Gamma(a)},$$

where $\Gamma(\cdot)$ is the gamma function.

In this exercise we will create a function that approximates the hypergeometric function by summing a finite number number of terms.

$$\tilde{F}(a, b; c; z) = \sum_{n=0}^{N} \frac{(a)_n (b)_n}{(c)_n} \cdot \frac{z^n}{n!}, \tag{1}$$

Problem definition

Create a function named `hypergeo` that takes as inputs the numbers a, b, c, z and N and computers the finite sum in equation (1).

Solution template

```
def hypergeo(a, b, c, z, N):
  #insert your code
  return F
```

Input

`a, b, c`	Parameters of the hypergeometric function (decimal numbers).
`z`	Function argument (decimal number).
`N`	Index of last term (positive whole number).

Output

`F`	Approximation of hypergeometric function (decimal number).

Example

Consider the following input: $a = 1.1$, $b = 2.2$, $c = 3.3$, $z = 0.5$, and $N = 3$. Thus we must sum the following 4 terms:

$$\begin{aligned}\tilde{F} &= \frac{1 \cdot 1}{1} \cdot \frac{z^0}{0!} + \frac{a \cdot b}{c} \cdot \frac{z^1}{1!} + \frac{a(a+1) \cdot b(b+1)}{c(c+1)} \cdot \frac{z^2}{2!} + \frac{a(a+1)(a+2) \cdot b(b+1)(b+2)}{c(c+1)(c+2)} \cdot \frac{z^3}{3!} \\ &= \frac{1 \cdot 1}{1} \cdot \frac{0.5^0}{0!} + \frac{1.1 \cdot 2.2}{3.3} \cdot \frac{0.5^1}{1!} + \frac{1.1(1.1+1) \cdot 2.2(2.2+1)}{3.3(3.3+1)} \cdot \frac{0.5^2}{2!} + \frac{1.1(1.1+1)(1.1+2) \cdot 2.2(2.2+1)(2.2+2)}{3.3(3.3+1)(3.3+2)} \cdot \frac{0.5^3}{3!} \\ &\approx 1 + 0.3666667 + 0.1432558 + 0.0586538 \approx \underline{1.5685763}\end{aligned}$$

Assignment E Name formatting

When formatting a list of references, it is often required that the author names are stated in a consistent manner. In this exercise we will create a function which takes in a full name in any of the formats specified below, and return the name formatted in a particular way.

We assume that the name given as input is either written with the family name first, followed by a comma, and then the given name(s), or as the given names(s) followed by the family name. Given names can either be written in full or abbreviated by their first letter. We can assume that there is either one or two given names. In case a given name is abbreviated, it may or may not be followed by a period. For example, the name `Evelyn Boyd Granville` can be written in one of the following 14 ways.

Family name+Comma+Given name(s)	Given name(s)+Family name
`Granville, Evelyn Boyd`	`Evelyn Boyd Granville`
`Granville, E. Boyd`	`E. Boyd Granville`
`Granville, Evelyn B.`	`Evelyn B. Granville`
`Granville, E. B.`	`E. B. Granville`
`Granville, E Boyd`	`E Boyd Granville`
`Granville, Evelyn B`	`Evelyn B Granville`
`Granville, E B`	`E B Granville`

The output of the function must be in the format of family name, followed by comma, folowed by given name(s) abbreviated by their initial without period. Thus, in the above example, the output must be:

```
Granville, E B
```

Note that in the input and output, the names are separated by a space character, and that there is a space character after the comma.

Problem definition

Create a function named `formatName` that takes as input a string containing a name formatted similar to one of the names in the table above, and returns the name formatted as specified above.

Solution template

```python
def formatName(nameIn):
  #insert your code
  return nameOut
```

Input	
`nameIn`	Input name (text string).

Output	
`nameOut`	Output name (text string).

Example

Given the name `Charles Babbage` as input, the function must return the string

```
Babbage, C
```

Given the name `Lovelace, A. Ada` as input, the function must return the string

```
Lovelace, A A
```