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-aug17/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) `rgb2hue.py`
   (b) `cardValidation.py`
   (c) `polygonPerimeter.py`
   (d) `rotateScale.py`
   (e) `costliestCar.py`

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

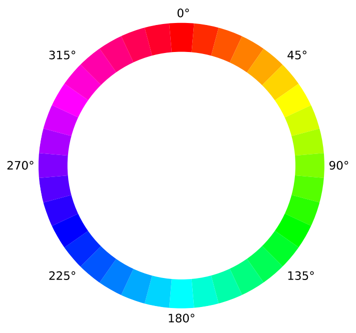
After the exam, your solutions will be automatically evaluated on CodeJudge on a range of different tests, to check that they work correctly in general. The assessment of your solution is based only on how many of the automated tests it passes.

- 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 Color hue

On a computer, colors are often represented in the RGB color model, i.e., as three numbers indicating the amount of red, green, and blue light. Another way to represent color is the HSL color model which consists of three numbers indicating the hue, saturation and luminance.

Given a color in RGB representation, the hue (H, measured in degrees) can be computed as described below. Let R, G, and B denote the RGB-values (in percent, represented by numbers between 0 and 1). We let Δ denote the range of the RGB-values (the difference between the maximum and minimum value),



$$\Delta = \max(R, G, B) - \min(R, G, B).$$

1. First, compute H according to the following formula:

$$H = \begin{cases} 60 \cdot \dfrac{G-B}{\Delta} & \text{If } R \text{ is the largest RGB-value, } R = \max(R,G,B). \\ 120 + 60 \cdot \dfrac{B-R}{\Delta} & \text{If } G \text{ is the largest RGB-value, } G = \max(R,G,B). \\ 240 + 60 \cdot \dfrac{R-G}{\Delta} & \text{If } B \text{ is the largest RGB-value, } B = \max(R,G,B). \end{cases}$$

   You can assume that $\Delta \neq 0$. Note that if two of the RGB-values are both maximum, the corresponding lines in the formula will lead to the same result.

2. If the computed value of H is negative, you must add 360 to get the correct value in degrees, so we have

$$0^\circ \leq H < 360^\circ.$$

Problem definition

Create a function named `rgb2hue` that takes as input the three RGB-values and computes the hue in degrees.

Solution template

```python
def rgb2hue(R, G, B):
  #insert your code
  return H
```

Input	
`R, G, B`	Red, green, and blue color value in percent (decimal number between 0 and 1).
Output	
`H`	Hue in degrees (decimal number, $0 \leq H < 360$).

Example

Consider the RGB-values $R = 0.6$, $G = 0.2$, $B = 0.3$. We can now compute

$$\Delta = \max(R, G, B) - \min(R, G, B) = 0.6 - 0.2 = 0.4.$$

Since the red color value is largest we compute

$$H = 60 \cdot \frac{G - B}{\Delta} = 60 \cdot \frac{0.2 - 0.3}{0.4} = -15$$

Because the result is negative, we must add 360, to yield the final result

$$H = -15 + 360 = \underline{345}$$

Assignment B Credit card validation

From a 16-digit credit card number, a checksum can be computed using the following algorithm (called the Luhn algorithm) that makes it possible to check whether or not the card number is valid :

1. Starting with the first digit from the left, convert *every other* digit (i.e. digit at position 1, 3, 5 etc.) according to this table :

Digit	0	1	2	3	4	5	6	7	8	9
Convert to	0	2	4	6	8	1	3	5	7	9

   The remaining digits are kept as they were.

2. Compute the checksum by adding all the digits together.

If the checksum is divisible by 10, the card number is valid.

Problem definition

Create a function named `cardValidation` that takes as input a 16-digit credit card number as a text string, and returns the checksum computed as described above.

Solution template

```
def cardValidation(cardnumber):
  #insert your code
  return checksum
```

Input

`cardnumber`	16-digit credit card number (text string).

Output

`checksum`	Checksum (whole number).

Example

For example, consider the credit card number `4024007156748096`. The first digit (4) is converted to 8; the second digit (0) is kept as it is; the third digit (2) is converted to 4; the fourth digit (4), is kept as it is; and so on. The checksum can thus be computed as:

Card number:	4	0	2	4	0	0	7	1	5	6	7	4	8	0	9	6
Convert every other digit:	8	↓	4	↓	0	↓	5	↓	1	↓	5	↓	7	↓	9	↓

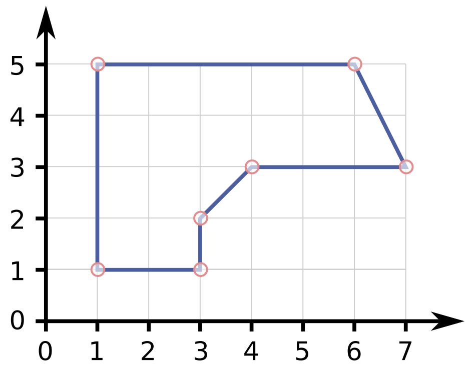
Add all digits: $8+0+4+4+0+0+5+1+1+6+5+4+7+0+9+6=\underline{60}$

The function must return the checksum 60 (and since it is divisible by 10 the card number is valid).

B

Assignment C Perimeter of a polygon

A polygon with n vertices can be specified by a list of vertex coordinates $\{(x_1, y_1),\ (x_2, y_2),\ \dots,\ (x_n, y_n)\}$. For example, the polygon shown in the figure below can be specified by the coordinates $(1, 1), (3, 1), (3, 2), (4, 3), (7, 3), ($



Based on the coordinates, the perimeter (circumference) of the polygon can be computed as the sum of the lengths of its sides, using the following formula

$$P = \sqrt{(x_1-x_2)^2 + (y_1-y_2)^2} + \sqrt{(x_2-x_3)^2 + (y_2-y_3)^2} + \cdots + \sqrt{(x_{n-1}-x_n)^2 + (y_{n-1}-y_n)^2} + \sqrt{(x_n-x_1)^2 + (y_n-y_1)^2}.$$

Note that the formula for the perimeter contains one term (Pythagoras' formula) for each edge in the polygon.

Problem definition

Create a function named `polygonPerimeter` that takes as input two vectors containing the x- and y-coordinates respectively, and returns the perimeter of the polygon. The function must work for polygons with any number of corners, $n \geq 3$.

Solution template

```
def polygonPerimeter(x, y):
  #insert your code
  return P
```

Input

`x, y`	X- and y-coordinates specifying a polygon (vectors of decimal numbers).

Output

`P`	Perimeter of the polygon.

Example

Consider the following input coordinates, which coorespond to the polygon in the figure.

$$x = [1,\ 3,\ 3,\ 4,\ 7,\ 6,\ 1], \quad y = [1,\ 1,\ 2,\ 3,\ 3,\ 5,\ 5]$$

The perimeter can the be computed as

$$\begin{aligned} P &= \sqrt{(x_1-x_2)^2 + (y_1-y_2)^2} + \sqrt{(x_2-x_3)^2 + (y_2-y_3)^2} + \sqrt{(x_3-x_4)^2 + (y_3-y_4)^2} + \\ &\qquad \sqrt{(x_4-x_5)^2 + (y_4-y_5)^2} + \sqrt{(x_5-x_6)^2 + (y_5-y_6)^2} + \sqrt{(x_6-x_7)^2 + (y_6-y_7)^2} + \sqrt{(x_7-x_1)^2 + (y_7-y_1)^2} \\ &= \sqrt{(1-3)^2 + (1-1)^2} + \sqrt{(3-3)^2 + (1-2)^2} + \sqrt{(3-4)^2 + (2-3)^2} + \\ &\qquad \sqrt{(4-7)^2 + (3-3)^2} + \sqrt{(7-6)^2 + (3-5)^2} + \sqrt{(6-1)^2 + (5-5)^2} + \sqrt{(1-1)^2 + (5-1)^2} \\ &= \sqrt{4} + \sqrt{1} + \sqrt{2} + \sqrt{9} + \sqrt{5} + \sqrt{25} + \sqrt{16} \ = \ 2 + 1 + \sqrt{2} + 3 + \sqrt{5} + 5 + 4 \ \approx \ \underline{18.650} \end{aligned}$$

Assignment D Rotation and scaling

A point (x, y) in a 2-d vector space can be rotated and scaled around a center point (a, b) according to this matrix/vector formula:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x-a \\ y-b \end{bmatrix} \cdot s + \begin{bmatrix} a \\ b \end{bmatrix},$$

where θ is the angle of rotation measured in radians, s is the scaling factor, and (x', y') are the resulting rotated and scaled coordinates.

Problem definition

Create a function named `rotateScale` that takes as input a $2 \times N$ matrix containing N pairs of x-, and y-coordinates, an angle θ, a scale factor s, as well as a center point (a, b), and returns a $2 \times N$ matrix containing the rotated and scaled coordinates. The function must work for any number of coordinates $N \geq 1$.

Solution template

```python
def rotateScale(coordinates, center, theta, scale):
  #insert your code
  return newCoordinates
```

Input	
`coordinates`	X- and y-coordinates of points to be rotated ($2 \times N$ matrix).
`center`	Center of rotation, i.e. point (a, b) in formula (vector).
`theta`	Angle of rotation, i.e. θ in formula (decimal number, in radians).
`scale`	Scale factor, i.e. s in formula (decimal number).
Output	
`newCoordinates`	X- and y-coordinates of the rotated and scaled points ($2 \times N$ matrix).

Example

We wish to rotate and scale the following seven points $(1, 1), (3, 1), (3, 2), (4, 3), (7, 3), (6, 5), (1, 5)$ around the center point $(a, b) = (3, 1)$ at an angle of $\theta = -\frac{\pi}{3}$ radians and a scale factor of $s = 2$. The points are collected in the following input matrix:

$$\texttt{coordinates}: \begin{bmatrix} 1 & 3 & 3 & 4 & 7 & 6 & 1 \\ 1 & 1 & 2 & 3 & 3 & 5 & 5 \end{bmatrix}.$$

After rotating and scaling each point according to the formula, we get the following result (shown rounded to three decimals):

$$\texttt{newCoordinates}: \begin{bmatrix} 1 & 3 & 4.732 & 7.464 & 10.464 & 12.928 & 7.928 \\ 4.464 & 1 & 2 & 1.268 & -3.928 & -0.196 & 8.464 \end{bmatrix},$$

where, for example, the rotated coordinates of the last (seventh) point $(x_7, y_7) = (1, 5)$ is computed as:

$$\begin{bmatrix} x_7' \\ y_7' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_7 - a \\ y_7 - b \end{bmatrix} \cdot s + \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \cos(-\frac{\pi}{3}) & -\sin(-\frac{\pi}{3}) \\ \sin(-\frac{\pi}{3}) & \cos(-\frac{\pi}{3}) \end{bmatrix} \begin{bmatrix} 1-3 \\ 5-1 \end{bmatrix} \cdot 2 + \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} -2 \\ 4 \end{bmatrix} \cdot 2 + \begin{bmatrix} 3 \\ 1 \end{bmatrix} = \begin{bmatrix} (-\frac{1}{2} \cdot 2 + \frac{\sqrt{3}}{2} \cdot 4) \cdot 2 + 3 \\ (\frac{\sqrt{3}}{2} \cdot 2 + \frac{1}{2} \cdot 4) \cdot 2 + 1 \end{bmatrix} \approx \begin{bmatrix} 7.928 \\ 8.464 \end{bmatrix}$$

Assignment E Costliest car within budget

When buying a new car, you can choose between different trim levels and add optional extras.

Base price
159 000

Trim level	Price
`Access`	0
`Comfort`	22 000
`Sport`	44 000

Extra options	Price
`Cruise control`	4 000
`Air conditioning`	8 000
`Alloy wheels`	13 000
`Chrome spoiler`	7 000

To configure a car you must pay the base price and you must choose a trim level. In addition, you can add any number of extra options (each extra option can not be added more than once).

We define a car specification as a string containing the trim level, and extra options as written in the tables above, separated by a comma and a space character, and given in the same order as they appear in the tables above. An example is the following specification string:

```
Comfort, Cruise control, Alloy wheels, Chrome spoiler
```

The price of this car would be: $159\,000 + 22\,000 + 4\,000 + 13\,000 + 7\,000 = \underline{205\,000}$

Problem definition

Create a function named `costliestCar` that takes as input a number `maxPrice` that specifies the maximum allowed price of the car. The function must return the specification of the costliest (most expensive) car with a total cost less than or equal to the maximum price (you may assume that the maximum price is greater than the base price and that there is a unique most expensive configuration). This can be done by computing the price of all possible combinations and then finding the most expensive configuration that does not exceed the budget: With 3 trim levels and 4 extra options, there are $3 \cdot 2^4 = 48$ different possible configurations. The specification must be returned as a string as defined above.

Solution template

```python
def costliestCar(maxPrice):
  #insert your code
  return carSpecification
```

Input	
`maxPrice`	Maximum price of car (whole number).
Output	
`carSpecification`	Car specification (string).

Example

Consider a maximum price of 170 500. Subtracting the base price, there is 11 500 left. The most expensive combination within the budget is the trim level `Access` with the extra options `Cruise control` and `Chrome spoiler`, so the output of the function must be:

```
Access, Cruise control, Chrome spoiler
```

The total price of the car is thus $159\,000 + 0 + 4\,000 + 7\,000 = 170\,000$, which does not exceed the budget.

E