Technical University of Denmark

Course name	Introduction to programming and data processing
Course number	02631, 02633, 02692
Aids allowed	All Aids
Exam duration	2 hours
Weighting	All exercises have equal weight

Contents

Assignment A: Space weight 2

Assignment B: Planet classification 3

Assignment C: 15-puzzle 4

Assignment D: Polygon convexity check 5

Assignment E: Sports medals 6

Submission details

You must hand in your solution electronically:

1. You can upload your solutions individually on CodeJudge (`dtu.codejudge.net/prog-f18/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) `weightOnPlanet.py`

   (b) `planets.py`

   (c) `inversions.py`

   (d) `polygon.py`

   (e) `medal.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 Space weight

Did you ever wonder how much you would weigh on another planet? The following table shows the weight on six planets, relative to the weight on earth.

Number	Planet	Relative weight (R)
1	Venus	0.91
2	Mars	0.38
3	Jupiter	2.53
4	Saturn	1.06
5	Uranus	0.89
6	Neptune	1.13

To compute the weight of a person on one of these planets, you simply multiply their weight on earth by the relative weight (R) for the planet.

Problem definition

Create a function named `weightOnPlanet` that takes as input the weight of a person on earth, and a planet number from the table above. The function must return the weight of the person on that planet.

Solution template

```
def weightOnPlanet(weight, planetNumber):
  #insert your code
  return newWeight
```

Input	
`weight`	Weight on earth (decimal number.)
`planetNumber`	Planet number, see table (whole number between 1 and 6.)

Output	
`newWeight`	Weight on the planet (decimal number.)

Example

Consider a person who weighs 82 kg and travels to Saturn (planet number 4). His weight on Saturn would be:

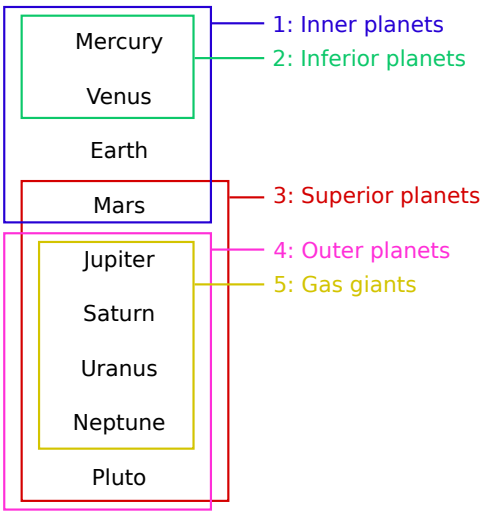
$$82 \cdot 1.06 = \underline{86.92}$$

A

Assignment B Planet classification

The planets in the solar system can be classified as belonging to one or more of the following categories:

1. Inner planets
2. Inferior planets
3. Superior planets
4. Outer planets
5. Gas giants



The figure shows which planets belong to each of the categories, and we will refer to each category by its number, 1–5.

Problem definition

Create a function named `planets` that takes as input a vector of numbers between 1 and 5, corresponding to a list of planet categories. The function must return an array of strings that contains the names of the planets that belong to one or more of the input categories. The order of the planet names must be as in the figure, and the planet names must be written with the first letter capitalized as in the figure.

Solution template

```
def planets(planetCategories):
  #insert your code
  return planetNames
```

Input	
`planetCategories`	List of planet category numbers (vector with whole numbers 1–5.)

Output	
`planetNames`	List of planet names (array of strings.)

Example

Consider the following list of planet categories as input:

4, 5, 2

These are the categories *4: Outer planets*, *5: Gas giants*, and *2: Inferior planets*. The planets that belong to one or more of these categories are (listed in the correct order):

`Mercury`, `Venus`, `Jupiter`, `Saturn`, `Uranus`, `Neptune`, `Pluto`

The final result must be an array of strings containing these planet names.

B

Assignment C 15-puzzle

The objective of the 15-puzzle is to slide 15 tiles numbered 1–15 on a 4-times-4 board (with one empty space) such that they are placed in correct numeric order.

If we are given a starting position, with the empty space at the lower right corner, the puzzle is only solvable if the number of *inversions* on the board is an even number. If we write out the numbers on the board as a single vector, row by row, the number of inversions is defined as the number of times a "larger" number is followed by a "smaller" number, i.e., a is followed by b where $a > b$.

Problem definition

Create a function named `inversions` that takes as input a vector containing 15 numbers (integers 1–15) corresponding to a configuration of a board for a game of 15-puzzle taken row by row. The function must return the number of inversions in the position.

```
def inversions(board):
  #insert your code
  return inv
```

Input

`board`	Board (vector with 15 elements, numbers 1–15.)

Output

`inv`	Number of inversions (whole number).

Example

Consider the board shown in the figure, which is represented by the following input vector:

`board`: 15, 2, 1, 12, 8, 5, 6, 11, 4, 9, 10, 7, 3, 14, 13

We can mark the inversions (transition from larger to smaller number) by a down arrow, and non-inversions by an up arrow.

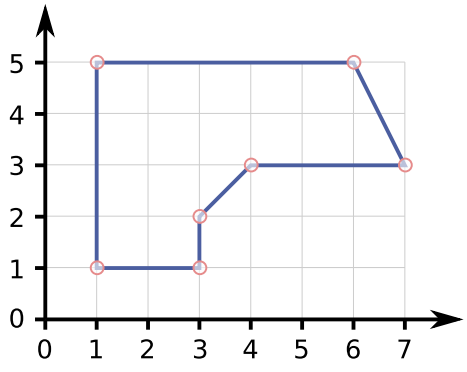
15↘2↘1↗12↘8↘5↗6↗11↘4↗9↗10↘7↘3↗14↘13

Counting the number of inversions, we arrive at the final result: 8.

C

Assignment D Polygon convexity check

A polygon can be specified by a list of vertex coordinates $\{(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)\}$. For example, the polygon shown in the figure below can be specified by the coordinates $\{(1,1), (3,1), (3,2), (4,3), (7,3), (6,5), (1,5)\}$.



A polygon is said to be convex when no line segment bewteen any two points on the boundary can ever go outside the polygon. We can examine if a polygon is convex, by going through each vertex of the polygon and checking that we keep "turning in the same direction".

For the k'th vertex in the polygon, we can define the following cross product:

$$c_k = (x_k - x_{k-1})(y_{k+1} - y_k) - (x_{k+1} - x_k)(y_k - y_{k-1})$$

where we use the following convention: $x_0 \equiv x_n$, $y_0 \equiv y_n$, $x_{n+1} \equiv x_1$, and $y_{n+1} \equiv y_1$ (cyclic indexing) where n is the number of vertices in the polygon. If c_k has the same sign, for all values of k, then the polygon is convex. In this exercise, we will simply compute the values c_1, c_2, $\ldots$, c_n.

Problem definition

Create a function named `polygon` that takes as input two vectors containing the x- and y-coordinates respectively, and returns the values c_1, c_2, $\ldots$, c_n as a vector.

Solution template

```
def polygon(x, y):
  #insert your code
  return C
```

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

Output	
`C`	Cross product for each vertex (vector of decimal numbers).

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 first cross product, c_1, can be computed as:

$$\begin{aligned} c_1 &= (x_1 - x_0)(y_2 - y_1) - (x_2 - x_1)(y_1 - y_0) \\ &= (x_1 - x_n)(y_2 - y_1) - (x_2 - x_1)(y_1 - y_n) \\ &= (1-1)(1-1) - (3-1)(1-5) = 8 \end{aligned}$$

In the same way, all the cross products can be computing, yielding the following result.

$$\texttt{C}\text{: } [8,\ 2,\ -1,\ -3,\ 6,\ 10,\ 20]$$

Assignment E Sports medals

In a sports competition for children, medals are awarded to the participants based on how many points they score and their age group according to the following table:

Medal	B (Baby)	P (Preschooler)	G (Gradeschooler)	T (Teen)
`gold`	10–20	15–20	17–20	19–20
`silver`	5–9	10–14	14–16	17–18
`bronze`	0–4	5–9	11–13	15–16
no medal	—	0–4	0–10	0–14

Number of points required to win the different medals.

Problem definition

Create a function named `medal` that takes as input an string representing one of the four age groups, as well as a point score. The function must only consider the first character in the input-string that defines the age group (i.e. `B`, `P`, `G`, or `T`). The function must return a text string which depends on which medal (if any) the participant has won.

If the participant has won a medal, the return value must be on the following form:

`You got` <pts> `points and won a` <mdl> `medal.`

Otherwise, if the participant did not win a medal, the return value must be on the following form:

`You got` <pts> `points.`

In these return values, the placeholders <pts> and <mdl> must respectively be replaced by the number of points and the type of medal (`gold`, `silver`, or `bronze`).

Solution template

```
def medal(ageGroup, points):
  #insert your code
  return message
```

Input	
`ageGroup`	Age group (string).
`points`	Number of points (whole number between 0 and 20.)

Output	
`message`	Message about points and medal (text string).

Example

Consider the following input:

`ageGroup: Preschooler` `points: 12.`

According to the table, a “Preschooler” with 10–14 points should get a silver medal, so the final result must be the string

`You got 12 points and won a silver medal.`

E