Mid-term

GRA4157

13th October 2022

1 Dictionaries and lists

Consider the list of integers (int).

```
l = [1, 2, -5, 3, 12, 20, 45, 13, -10, 21]
```

a) Write a function that takes such a list as input argument, loops over the list and then returns the greatest value in the list. (built-in max function not allowed).

b) What will be the outputs of the following calls?

```
print(l[1:3]+l[4:6])
print(l[3]+l[6])
print(2*l[8])
print(2*l[8:])
```

c) Consider the dictionary

```
d = {'Oslo':2.5, 'London':12.3, 'Paris':11.0, 9.0: 'Berlin'}
```

Someone made a mistake when creating this dictionary. Your task is to correct the mistake so the format is consistent. You may update the dictionary d or create a new one. Your code should be generalized such that all potential mistakes where the order is wrong between key:value are fixed.

2 Reading from and writing to file

Consider the file input.txt:

```
This is a header
This is a header
Numbers: 1 2 3
Numbers: 2 3 4
5 6 7
```

The numbers in this file could be organized as a 3x3 matrix.

a) Read the file using pure python and for each row, put all numbers found in the row in a list. If no number is found on a line, no action is required. Call the lists row1, row2 and row3.

As an example, row 1 should end up as [1,2,3]

b) Read the file using pure python and put numbers in each column in three lists. Call the lists col1, col2 and col3. If no number is found on a line, no action is required.

As an example, col 1 should end up as [1,2,5]

c) Write a program that writes the 3x3 matrix to a file "out.csv". The file should have the header "a,b,c" and the numbers in each row of the matrix should be separated by commas. You can assume that you have access to the rows and column lists from a) and b).

3 Vectorized computations

a) Explain what is wrong with the following code:

```
import numpy as np

def f(x):
    if x > 0:
        return x
    else:
        return -x

x = np.linspace(-1,1,5)
print(f(x))
```

b) Modify the function f, such that the program above works and allows for vectorized computations of f (e.g. uses numpy calculations instead of python for loops).

c) You have been hired as a consultant to speed up computations for Price-control AS. Assume that you have already read 100 million high resolution time and price values (t,p) for a given commodity. The company previously used the following code to compute the instantaneous change in price over time:

```
#t, p already read in from file
n = len(p)
d = []
for i in range(n-1):
    dt = t[i+1] - t[i]
    di = (p[i+1] - p[i])/dt
```

```
d.append(di)
```

Re-write the program to use vectorized computations with numpy instead of a python for-loop. Also, plot d versus t with matplotlib. The x axis should be labeled "time [s]" and the y axis should be labeled "Price change/s".

4 Pandas and web scraping

Assume we have the csv-file out.csv:

```
a,b,c
1,2,3
2,3,4
5,6,7
```

a) Write a program that 1) reads the file into a pandas DataFrame. 2) extracts numbers from the columns into three pandas Series, a, b and c. 3) plots a versus b.

In the following, we will consider the wikipedia page with a list of Norwegian footballers (https://en.wikipedia.org/wiki/List_of_Norway_international_footballers). Table number 2 (if we start counting at 1) in the wikipedia page has the following structure:

Norway national team football players with at least 25 caps

No.	Player	Pos.	Caps	Goals	Debut		Last or most recent match		Ref.
					Date	Opponent	Date	Opponent	
1	John Arne Riise	DF / MF	110	16	31 January 2000	Iceland	22 March 2013	Albania	[2]
2	Thorbjørn Svenssen	DF	104	0	11 June 1947	Poland	16 May 1962	Netherlands	[3]
3	Henning Berg	DF	100	9	13 May 1992	Faroe Islands	27 May 2004	Wales	[4]
4	Erik Thorstvedt	GK	97	0	13 November 1982	Kuwait	27 March 1996	Northern Ireland	[5]
5	John Carew	FW	91	24	18 November 1998	Egypt	11 October 2011	Cyprus	[6]
=	Brede Hangeland	DF	91	4	20 November 2002	Austria	27 May 2014	France	[7]
7	Øyvind Leonhardsen	MF	86	19	31 October 1990	Cameroon	7 June 2003	Denmark	[8]
8	Kjetil Rekdal	MF	83	17	28 May 1987	Italy	27 May 2000	Slovakia	[9]
=	Morten Gamst Pedersen	MF	83	17	18 February 2004	Northern Ireland	9 September 2014	Italy	[10]

Figure 1: The first 9 rows of the table with Norweigan footballers

b) 1) Read the web page (HTML) into a list of DataFrames with pandas, and extract the table above as a DataFrame in a new variable named "players".

2) Use sort_values (Attachment 3 below) to sort the DataFrame by number of goals and print the names of the top 10 goal scorers of all time.

c) You will now build a team of the most scoring players in Norwegian football history. The team will thus consist of players from different time periods. The team should have a formation 4-4-2 which means it consists of 11 players

distributed in the following positions (Pos.): 1 goalkeeper (GK), 4 defenders (DF, FB), 4 midfielders (MF, HB) and 2 forwards (FW), see Figure 2. You can assume that you have access to the DataFrame that has the table above sorted by "Goals" at hand. **Note:** If a player has two positions, you are free to choose which of the two positions the player is considered for in the team draft, such that each player is only associated with one (pre-determined) position.

Your task is to print the names of the 11 players included in the team that will maximize the combined amount of goals scored under the constraint that they all play in their listed positions.

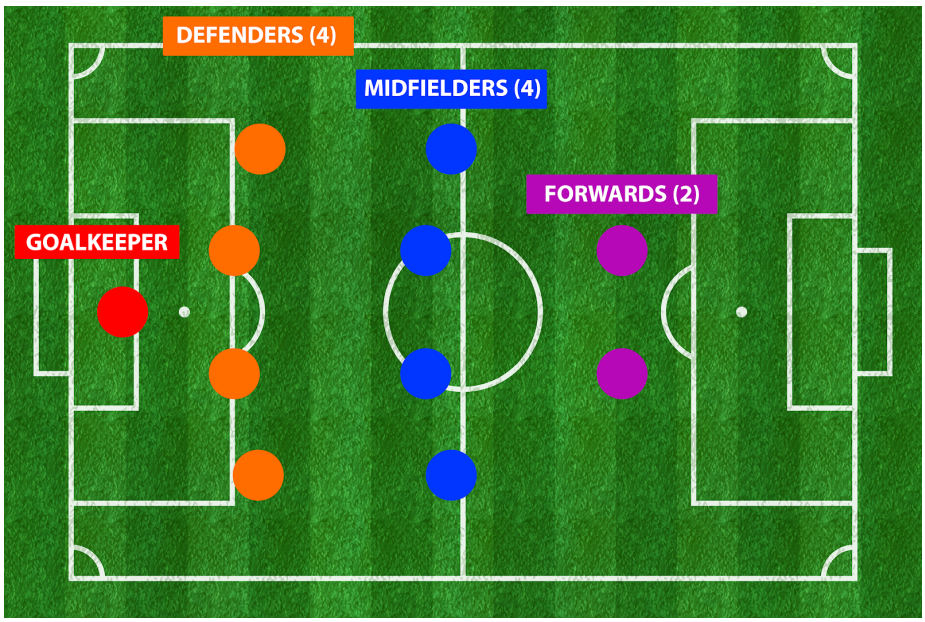
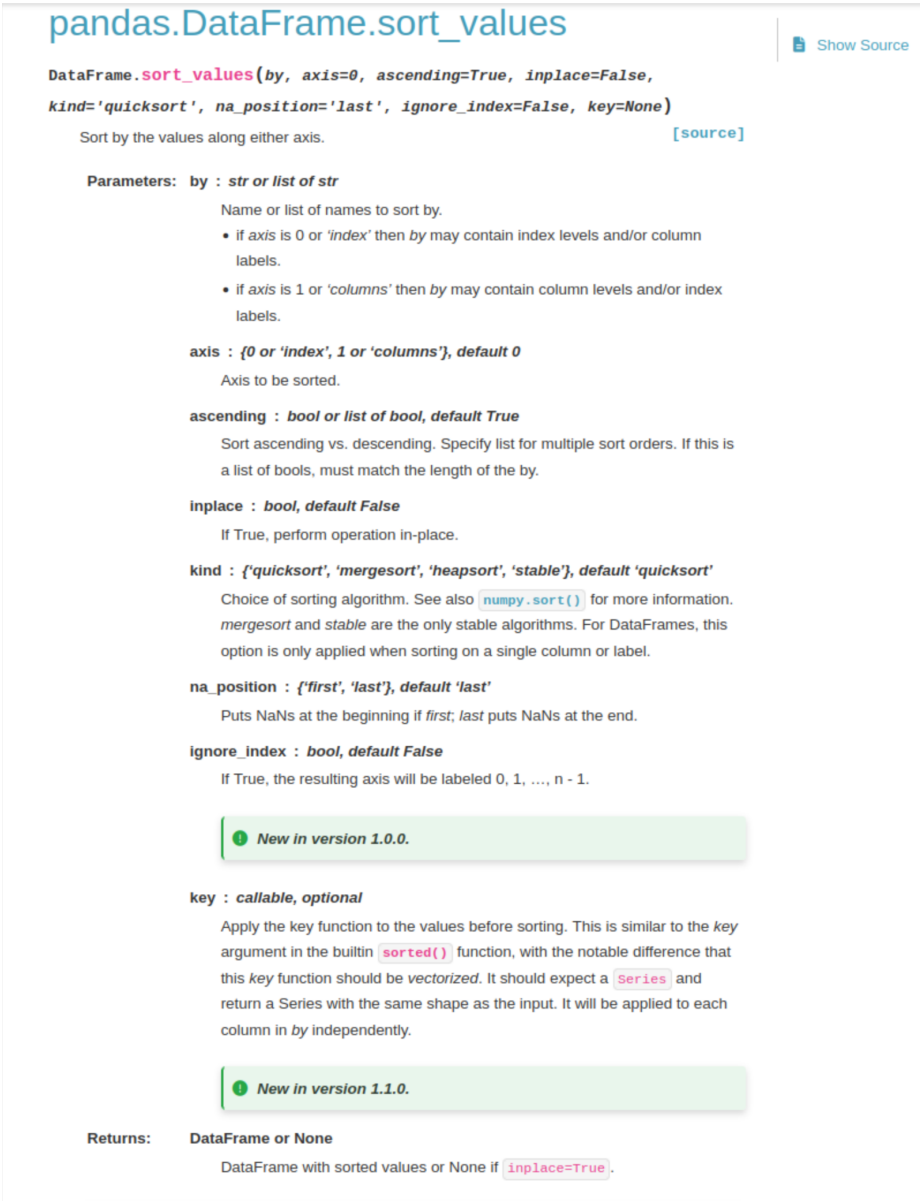


Figure 2: The 4–4–2 formation in football



pandas.DataFrame.sort_values

Show Source

`DataFrame.sort_values(by, axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last', ignore_index=False, key=None)` [source]

Sort by the values along either axis.

Parameters:

by : *str or list of str*

Name or list of names to sort by.

- if *axis* is 0 or *'index'* then *by* may contain index levels and/or column labels.
- if *axis* is 1 or *'columns'* then *by* may contain column levels and/or index labels.

axis : *{0 or 'index', 1 or 'columns'}, default 0*

Axis to be sorted.

ascending : *bool or list of bool, default True*

Sort ascending vs. descending. Specify list for multiple sort orders. If this is a list of bools, must match the length of the by.

inplace : *bool, default False*

If True, perform operation in-place.

kind : *{'quicksort', 'mergesort', 'heapsort', 'stable'}, default 'quicksort'*

Choice of sorting algorithm. See also `numpy.sort()` for more information. *mergesort* and *stable* are the only stable algorithms. For DataFrames, this option is only applied when sorting on a single column or label.

na_position : *{'first', 'last'}, default 'last'*

Puts NaNs at the beginning if *first*; *last* puts NaNs at the end.

ignore_index : *bool, default False*

If True, the resulting axis will be labeled 0, 1, …, n - 1.

New in version 1.0.0.

key : *callable, optional*

Apply the key function to the values before sorting. This is similar to the *key* argument in the builtin `sorted()` function, with the notable difference that this *key* function should be *vectorized*. It should expect a `Series` and return a Series with the same shape as the input. It will be applied to each column in *by* independently.

New in version 1.1.0.

Returns:

DataFrame or None

DataFrame with sorted values or None if `inplace=True`.

Figure 3: Attachment: DataFrame.sort_values