```
import numpy as np
mylist = list('abcedfghijklmnopqrstuvwxyz')
myarr = np.arange(26)
mydict = dict(zip(mylist, myarr))
```

1. Create a pandas series from each of the items below: a list, numpy and a dictionary.

```
mylist = list('abcedfghijklmnopqrstuvwxyz')
myarr = np.arange(26)
mydict = dict(zip(mylist, myarr))
ser = pd.Series(mydict)
```

2. Convert the series ser into a dataframe with its index as another column on the dataframe.

```
import numpy as np
ser1 = pd.Series(list('abcedfghijklmnopqrstuvwxyz'))
ser2 = pd.Series(np.arange(26))
```

3. Combine ser1 and ser2 to form a dataframe.

```
ser = pd.Series(list('abcedfghijklmnopqrstuvwxyz'))
```

4. Give a name to the series ser calling it ‘alphabets’.

```
ser1 = pd.Series([1, 2, 3, 4, 5])
ser2 = pd.Series([4, 5, 6, 7, 8])
```

5. From ser1 remove items present in ser2.
6. Get all items of ser1 and ser2 not common to both.

```
ser = pd.Series(np.random.normal(10, 5, 25))
```

7. Compute the minimum, 25th percentile, median, 75th, and maximum of ser.

```
ser = pd.Series(np.take(list('abcdefgh'), np.random.randint(8,
```

```
                                size=30)))
```

8. Calculte the frequency counts of each unique value ser.

```
np.random.RandomState(100)
ser = pd.Series(np.random.randint(1, 5, [12]))
```

9. From ser, keep the top 2 most frequent items as it is and replace everything else as ‘Other’.

```
ser = pd.Series(np.random.random(20))
```

10. Bin the series ser into 10 equal deciles and replace the values with the bin name.

```
ser = pd.Series(np.random.randint(1, 10, 35))
```

11. Reshape the series ser into a dataframe with 7 rows and 5 columns.

```
ser = pd.Series(np.random.randint(1, 10, 7))
```

12. Find the positions of numbers that are multiples of 3 from ser.

```
ser = pd.Series(list('abcdefghijklmnopqrstuvwxyz'))
pos = [0, 4, 8, 14, 20]
```

13. From ser, extract the items at positions in list pos.

```
ser1 = pd.Series(range(5))
ser2 = pd.Series(list('abcde'))
```

14. Stack ser1 and ser2 vertically and horizontally (to form a dataframe).

```
ser1 = pd.Series([10, 9, 6, 5, 3, 1, 12, 8, 13])
```

```
ser2 = pd.Series([1, 3, 10, 13])
```

15. Get the positions of items of ser2 in ser1 as a list.

```
truth = pd.Series(range(10))
pred = pd.Series(range(10)) + np.random.random(10)
```

16. Compute the mean squared error of truth and pred series.

```
ser = pd.Series(['how', 'to', 'kick', 'ass?'])
```

17. Change the first character of each word to upper case in each word of ser.

```
ser = pd.Series(['how', 'to', 'kick', 'ass?'])
```

18. How to calculate the number of characters in each word in a series?

```
ser = pd.Series(['01 Jan 2010', '02-02-2011', '20120303', '2013/04/04', '2014-05-05', '2015-06-06T12:20'])
```

19. How to convert a series of date-strings to a timeseries?

```
ser = pd.Series(['01 Jan 2010', '02-02-2011', '20120303', '2013/04/04', '2014-05-05', '2015-06-06T12:20'])
```

20. Get the day of month, week number, day of year and day of week from ser.

```
ser = pd.Series(['Jan 2010', 'Feb 2011', 'Mar 2012'])
```

21. Change ser to dates that start with 4th of the respective months.

```
ser = pd.Series(['Apple', 'Orange', 'Plan', 'Python', 'Money'])
```

22. From ser, extract words that contain atleast 2 vowels.

```
emails = pd.Series(['buying books at amazom.com', 'rameses@egypt.com', 'matt@t.co',
'narendra@modi.com'])
pattern ='[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,4}'
```

23. Extract the valid emails from the series emails. The regex pattern for valid emails is provided as reference.

```
fruit = pd.Series(np.random.choice(['apple', 'banana', 'carrot'], 10))
weights = pd.Series(np.linspace(1, 10, 10))
```

24. Compute the mean of weights of each fruit.

```
p = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
q = pd.Series([10, 9, 8, 7, 6, 5, 4, 3, 2, 1])
```

25. Compute the euclidean distance between series (points) p and q,

```
ser = pd.Series([2, 10, 3, 4, 9, 10, 2, 7, 3])
```

26. Get the positions of peaks (values surrounded by smaller values on both sides) in ser.