Congratulations! You passed!

Grade received 100%
Latest Submission Grade 100%
To pass 80% or higher

Go to next item

1. Consider the given NumPy arrays **a** and **b**. What will be the value of **c** after the following code is executed? **1 / 1 point**

```
import numpy as np

a = np.arange(8)
b = a[4:6]
b[:] = 40
c = a[4] + a[6]
```

46

Correct
Since array slices are passed by reference, the values at indices 4 and 5 in **a** will also be modified after the values in the array slice **b** are changed to 40. Hence, after the code is run, **a** = array([0, 1, 2, 3, 40, 40, 6, 7]). As **a[4]** = 40 and **a[6]** = 6, **c** = 40 + 6 = 46.

2. Given the string **s** as shown below, which of the following expressions will be **True**? **1 / 1 point**

```
import re
s = 'ABCAC'
```

○

```
re.match('A', s) == True

```

○

```
1  len(re.split('A', s)) == 2
```

○

```
1  len(re.search('A', s)) == 2
```

◉

```
1  bool(re.match('A', s)) == True
```

Correct
re.match('A', s) return an re.match object. Since it will find a match for 'A' in 's', thus converting this object to bool will return True.

3. Consider a string s. We want to find all characters (other than A) which are followed by triple A, i.e., have AAA to the right. We don't want to include the triple A in the output and just want the character immediately preceding AAA . Complete the code given below that would output the required result. **1 / 1 point**

```
def result():
    s = 'ACAABAACAAABACDBADDDFSDDDFFSSSASDAFAAACBAAAFASD'

    result = []
    # compete the pattern below
    pattern = "(\w)(?=[A]{3})"
    for item in re.finditer(pattern, s):
      # identify the group number below.
      result.append(item.group())

    return result

result()
```

Run
Reset

```
['C', 'F', 'B']
```

Correct
Good job!

4. **1 / 1 point**

```
>>> df
d    4
b    7
a   -5
c    3
dtype: int64
```

Consider the following 4 expressions regarding the above pandas Series **df**. All of them have the same value except one expression. Can you identify which one it is?

- ○
```
1  df['d']
```
- ○
```
1  df[0]
```
- ○
```
1  df.iloc[0]
```
- ◉
```
1  df.index[0]
```

✓ **Correct**
df.index[0] = 'd', which is different from the other three which are equal to 4

5. **1 / 1 point**

```
>>> s1
Mango          20
Strawberry     15
Blueberry      18
Vanilla        31
dtype: int64
```

```
>>> s2
Strawberry     20
Vanilla        30
Banana         15
Mango          20
Plain          20
dtype: int64
```

Consider the two pandas Series objects shown above, representing the no. of items of different yogurt flavors that were sold in a day from two different stores, **s1 and s2**. Which of the following statements is **True** regarding the Series **s3** defined below?

```
s3 = s1.add(s2)
```

- [x]
```
s3['Mango'] >=  s1.add(s2, fill_value = 0)['Mango']
```

- [ ]
```
s3['Blueberry'] == s1['Blueberry']
```

- [ ]
```
s3['Blueberry'] == s1.add(s2, fill_value = 0)['Blueberry']
```

```
s3['Plain'] >= s3['Mango']
```

Correct
This is True because the values of both are the same. Notice that the comparison type is greater than equal to and not strictly greater than.

6. In the following list of statements regarding a DataFrame df, one or more statements are correct. Can you identify all the correct statements? **1 / 1 point**

- [x] Every time we call df.set_index(), the old index will be discarded.

Correct

- [ ] Every time we call df.set_index(), the old index will be set as a new column.
- [ ] Every time we call df.reset_index(), the old index will be discarded.
- [x] Every time we call df.reset_index(), the old index will be set as a new column.

Correct

7. Consider the Series object S defined below. Which of the following is an **incorrect** way to slice S such that we obtain all data points corresponding to the indices 'b', 'c', and 'd'? **1 / 1 point**

```
S = pd.Series(np.arange(5), index=['a', 'b', 'c', 'd', 'e'])
```

```
S['b':'e']
```

○

```
S[['b', 'c', 'd']]
```

○

```
S[S <= 3][S > 0]
```

○

```
S[1:4]
```

✓ **Correct**

This will incorrectly extract all the data for rows, 'b', 'c', 'd', and 'e'. We only need the data for 'b', 'c' and 'd'. Remember, slicing with labels behaves differently than normal Python slicing in that the endpoint is inclusive.

8. **1 / 1 point**

```
>>> df
     a   b   c
R1   5   6  20
R2   5  82  28
R3  71  31  92
R4  67  37  49
```

Consider the DataFrame df shown above with indexes 'R1', 'R2', 'R3', and 'R4'. In the following code, a new DataFrame df_new is created using df. What will be the value of **df_new[1]** after the below code is executed?

```
1  f = lambda x: x.max() + x.min()
2  df_new = df.apply(f)
```

88

Correct

The lambda function, when applied to df, will output a Series object with the sum of maximum and minimum values of each column 'a', 'b', and 'c'. df1[1] refers to the second row of df1, which has index 'b'. The maximum and minimum values of column 'b' are 82 and 6 respectively. Hence the correct answer is 82 + 6 = 88.

9. **1 / 1 point**

	mean					amax				
Rank_Level	First Tier Top Unversity	Other Top Unversity	Second Tier Top Unversity	Third Tier Top Unversity	All	First Tier Top Unversity	Other Top Unversity	Second Tier Top Unversity	Third Tier Top Unversity	All
country										
Argentina	NaN	44.672857	NaN	NaN	44.672857	NaN	45.66	NaN	NaN	45.66
Australia	47.9425	44.645750	49.2425	47.285000	45.825517	51.61	45.97	50.40	47.47	51.61
Austria	NaN	44.864286	NaN	47.066667	45.139583	NaN	46.29	NaN	47.78	47.78
Belgium	51.8750	45.081000	49.0840	46.746667	47.011000	52.03	46.21	49.73	47.14	52.03
Brazil	NaN	44.499706	49.5650	NaN	44.781111	NaN	46.08	49.82	NaN	49.82

Consider the DataFrame named new_df shown above. Which of the following expressions will output the result (showing the head of a DataFrame) below?

	country	Argentina	Australia	Austria	Belgium	Brazil	Bulgaria	Canada	Chile	China	Colombia	...	Switzerland	Taiwan	Thai
	Rank_Level														
mean	First Tier Top Unversity	NaN	47.942500	NaN	51.875000	NaN	NaN	53.633846	NaN	53.592500	NaN	...	54.005000	54.210000	
	Other Top Unversity	44.672857	44.645750	44.864286	45.081000	44.499706	44.335	44.760541	44.7675	44.564267	44.4325	...	44.625000	44.476667	44.8
	Second Tier Top Unversity	NaN	49.242500	NaN	49.084000	49.565000	NaN	49.218182	NaN	47.868000	NaN	...	48.184000	NaN	
	Third Tier Top Unversity	NaN	47.285000	47.066667	46.746667	NaN	NaN	46.826364	NaN	46.926250	NaN	...	47.930000	47.065000	46.5
	All	44.672857	45.825517	45.139583	47.011000	44.781111	44.335	47.359306	44.7675	44.992575	44.4325	...	51.208846	45.012391	45.1

- ○ new_df.stack()
- ○ new_df.unstack()
- ○ new_df.stack().stack()
- ◉ new_df.unstack().unstack()

✓ **Correct**

When we unstack the new_df first time, we will get a Series with triple row index, when we unstack this Series again, the inner most row, which is actually the previous country, will be moved to the column index again.

10. **1 / 1 point**

	Item	Store	Quantity sold
0	item_1	A	10.0
1	item_1	B	20.0
2	item_1	C	NaN
3	item_2	A	5.0
4	item_2	B	10.0
5	item_2	C	15.0

Consider the DataFrame **df** shown above. What will be the output (rounded to the nearest integer) when the following code related to **df** is executed:

```
df.groupby('Item').sum().iloc[0]['Quantity sold']
```

30

Correct

Groupby will create two groups from this df which correspond to item_1 and item_2. Calling sum() will add the quantities sold for each item across all 3 stores (A, B and C). iloc[0] will get the first row, which corresponds to item_1 and the sum of 'Quantity sold' values for item_1 will be calculated ignoring NaN, which comes out to be 30.