Congratulations! You passed!

Grade received 100% **To pass** 80% or higher

Go to next item

You have new skill scores!

Great job! Keep learning and making progress in your courses to increase your skill scores.

Computer Programming	278	Data Structures	112
Mathematics	86	Mathematical Theory & Analysis	126
Programming Principles	229	Python Programming	249
Statistical Programming	202	Theoretical Computer Science	167

Final Quiz

Latest Submission Grade 100%

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.array([1])
b = a[:1]
b[0] = 4
c = a[:1 + a[:1]
```

- 4[]
- Correct
  Since array slices are passed by reference, the values at indices 0 and 1 in **a** will also be modified after the values in the array slice **b** are changed to 4[]. Hence, after the code is run, **a** = array([4, 1, 2, 3.45, 4[], 7]) and **c**[0] = 4 and **c**[1] = 4 + 4[] = 4[].

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

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

- `len(re.split('A', s)) == 2`
- `len(re.search('A', s)) == 2`
- `bool(re.match('A', s)) == True`
- `re.match('A', s) == True`

- Correct
  re.match('A', s) return an re.match object. Since it will find a match for 'A' in 'A', 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):
    s = 'ACABAACAAABACBAADDCFQDDFFFCCCCDDFAAACBAAAFADD'

    result = []
    # compute the pattern below
    pattern = 'r"(?=(?:(?:[A-D]
    for m in re.finditer(pattern, s):
    # identify the group number below
    result.append(m.group(1))

    return result
result()
```

Run

Reset

- 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?

- `df.iloc[0]`
- `df.index[0]`
- `df[0]`
- `df[ 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)
```

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

- 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[ b:e ]`
- `S[ b:e:1 ]`
- `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 indices '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[0]** after the below code is executed?

```
df = lambda x: x.abs() + x.std()
df_new = df.apply(df)
```

- 80

- 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'. df[0] 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					price					
Rank Level	First Tier Top University	Other Top University	Second Tier Top University	Third Tier Top University	All	First Tier Top University	Other Top University	Second Tier Top University	Third Tier Top University	All
country										
Argentina	NaN	44.673957	NaN	NaN	44.673957	NaN	45.66	NaN	NaN	45.66
Australia	47.7625	44.840156	45.5325	47.290000	45.850417	51.61	43.87	50.45	45.67	47.41
Austria	NaN	44.864286	NaN	47.390667	45.138462	NaN	44.29	NaN	NaN	47.78
Belgium	51.8100	45.061905	45.0842	46.196667	47.011905	52.33	46.21	46.73	45.14	50.33
Brazil	NaN	44.499756	49.5652	NaN	44.781111	NaN	46.08	49.82	NaN	48.82
...	...	...	...	...	...	...	...	...	...	...
Switzerland	NaN	54.000000	54.210000	NaN	54.105000	NaN	55.00	54.00	NaN	54.50
Thailand	NaN	44.000000	44.000000	44.000000	44.000000	NaN	44.00	44.00	NaN	44.00
Turkey	41.675667	44.845700	44.864250	45.061818	44.406786	44.335	44.79054	44.78375	44.564267	44.4325
United Arab Emirates	NaN	49.242500	NaN	49.084000	49.300000	NaN	49.210182	51.460833	NaN	49.194000
United Kingdom	47.290000	47.000667	46.748667	NaN	46.923556	NaN	46.929210	NaN	47.000000	47.000000
United States	46.675667	45.820217	45.138583	47.201100	44.797111	44.335	47.200508	44.78375	44.882675	44.4325
Uruguay	NaN	44.000000	NaN	NaN	44.000000	NaN	44.00	NaN	NaN	44.00
Venezuela	NaN	44.000000	NaN	NaN	44.000000	NaN	44.00	NaN	NaN	44.00
Vietnam	NaN	44.000000	NaN	NaN	44.000000	NaN	44.00	NaN	NaN	44.00
All	46.675667	45.820217	45.138583	47.201100	44.797111	44.335	47.200508	44.78375	44.882675	44.4325

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	Thailand	Tur
	Rank Level														
mean	First Tier Top University	NaN	47.675667	NaN	51.810000	NaN	NaN	53.450000	NaN	50.500769	NaN	...	54.000000	54.210000	
	Other Top University	44.673957	44.840700	44.864286	45.061905	44.499756	44.335	44.79054	44.78375	44.564267	44.4325	...	54.000000	54.478667	44.8
	Second Tier Top University	NaN	45.242500	NaN	49.084000	49.300000	NaN	49.210182	51.460833	NaN	49.194000		NaN		
	Third Tier Top University	47.290000	47.000667	47.390667	46.748667	NaN	NaN	46.929210	NaN	46.929210	NaN	...	47.000000	47.000000	46.9
	All	44.673957	45.820217	45.138583	47.201100	44.797111	44.335	47.200508	44.78375	44.882675	44.4325	...	51.208846	45.012381	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')[Quantity sold']
```

- 30

- Correct
  Grouping 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.