# Python Programming Quiz (version 1.2.3):

|  |  |
| --- | --- |
| **Name** | Supasun khumpraphan |
| **Date** | 15/08/2019 |

Please solve the problem using Python and comment the code to explain concept of your solution.

(Optional: you can solve by using Jupyter)

## Problem 1: Find the collection of the longest order sequence in any input.

**Example:**

|  |  |  |
| --- | --- | --- |
| **#** | **Input** | **Output** |
| 1 | "**124**01021" | ["124"] |
| 2 | "7**1245**011**1579**3" | ["1245", "1579"] |
| 3 | “15235137835692838387” | ['1378', '3569'] |

**FindLongestOrderSequence Function:**

|  |
| --- |
| def FindLongestOrderSequence(input):  # Variable declaration  temp, output = input[0], []  # Number from zeto to the end  for i in range(len(input) - 1):  # check value less than  if input[i] < input[i+1]:  temp += input[i+1]  # check value more than  else:  if temp:  output.append(temp)  temp = input[i+1]  # The most length  maxLen = max([len(el) for el in output])  # output my list  output = [val for val in output if len(val) == maxLen]  return output  print(FindLongestOrderSequence('12401021')) # ['124']  print(FindLongestOrderSequence('7124501115793')) # ['1245', '1579']  print(FindLongestOrderSequence('15235137835692838387')) # ['1378', '3569'] |

## Problem 2: Using Pandas to solve below questions

*Note: Fill your code in the provided section (optional: attached your Jupyter Notebook)*

**DataFrame: dfStudent**

|  |  |  |
| --- | --- | --- |
| **STUDENT\_ID (INDEX)** | **NAME** | **GENDER** |
| 10001 | Chris Evans | M |
| 10002 | Brie Larson | F |
| 10003 | Tom Holland | M |
| 10004 | Mark Ruffalo | M |
| 10005 | Scarlett Johansson | F |

**DataFrame: dfCourse**

|  |  |
| --- | --- |
| **COURSE\_ID (INDEX)** | **NAME** |
| X0001 | Database |
| X0002 | Programming |

**DataFrame: dfGrade**

|  |  |  |
| --- | --- | --- |
| **STUDENT\_ID (INDEX)** | **COURSE\_ID (INDEX)** | **SCORE** |
| 10001 | X0001 | 90 |
| 10001 | X0002 | 70 |
| 10002 | X0001 | 85 |
| 10002 | X0002 | 80 |
| 10003 | X0001 | 69 |
| 10003 | X0002 | 65 |
| 10004 | X0001 | 85 |
| 10005 | X0002 | 95 |

### 1. Create new columns in dfStudent

* FIRSTNAME: Substring NAME column before whitespace
* LASTNAME: Substring NAME column after whitespace
* DISPLAYNAME: First character of FIRSTNAME with “.” And LASTNAME

**Example:**

|  |  |  |  |
| --- | --- | --- | --- |
| **NAME** | **FIRSTNAME** | **LASTNAME** | **DISPLAYNAME** |
| Chris Evans | Chris | Evans | C. Evans |

**Code:**

|  |
| --- |
| import pandas as pd  # create DataFrame name dfStudent  dfStudent = pd.DataFrame({  'STUDENT\_ID':['10001','10002','10003','10004','10005'],  'NAME':['Chris Evans','Brie Larson','Tom Holland','Mark Ruffalo','Scarlett Johansson'],  'GENDER':['M','F','M','M','F']},  )  # index = STUDENT\_ID  dfStudent.set\_index('STUDENT\_ID',inplace=True)  # create DataFrame name dfCourse  dfCourse = pd.DataFrame({  'COURSE\_ID':['X0001','X0002'],  'NAME':['Database','Programming']  })  # index = COURSE\_ID  dfCourse.set\_index('COURSE\_ID',inplace=True)  # create DataFrame name dfGrade  dfGrade = pd.DataFrame({  'STUDENT\_ID':['10001','10001','10002','10002','10003','10003','10004','10005'],  'COURSE\_ID':['X0001','X0002','X0001','X0002','X0001','X0002','X0001','X0002'],  'SCORE':[90,70,85,80,69,65,85,95]  })  # index = STUDENT\_ID and COURSE\_ID  dfGrade.set\_index(['STUDENT\_ID','COURSE\_ID'],inplace=True)  # create columns FIRSTNAME,LASTNAME,DISPLAYNAME  dfStudent['FIRSTNAME'] = dfStudent['NAME'].apply(lambda x: x.split(' ')[0])  dfStudent['LASTNAME'] = dfStudent['NAME'].apply(lambda x: x.split(' ')[1])  dfStudent['DISPLAYNAME'] = dfStudent['NAME'].apply(lambda x: x[:1]) + '. ' + dfStudent['NAME'].apply(lambda x: x.split(' ')[1])  dfStudent |

### 2. Create a new column in dfGrade

- GRADE: follow the condition:

|  |
| --- |
| IF SCORE >= 80 THEN GRADE = A  ELSE IF SCORE >= 70 THEN GRADE = B  ELSE IF SCORE >= 60 THEN GRADE = C  ELSE GRADE = F |

**Example:**

|  |  |  |  |
| --- | --- | --- | --- |
| **STUDENT\_ID (INDEX)** | **COURSE\_ID (INDEX)** | **SCORE** | **GRADE** |
| 10001 | X0001 | 90 | A |

**Code:**

|  |
| --- |
| import pandas as pd  # create dfStudent  dfStudent = pd.DataFrame({  'STUDENT\_ID':['10001','10002','10003','10004','10005'],  'NAME':['Chris Evans','Brie Larson','Tom Holland','Mark Ruffalo','Scarlett Johansson'],  'GENDER':['M','F','M','M','F']},  )  # index dfStudent = STUDENT\_ID  dfStudent.set\_index('STUDENT\_ID',inplace=True)  # create dfCourse  dfCourse = pd.DataFrame({  'COURSE\_ID':['X0001','X0002'],  'NAME':['Database','Programming']  })  # index dfCourse = COURSE\_ID  dfCourse.set\_index('COURSE\_ID',inplace=True)  # create dfGrade  dfGrade = pd.DataFrame({  'STUDENT\_ID':['10001','10001','10002','10002','10003','10003','10004','10005'],  'COURSE\_ID':['X0001','X0002','X0001','X0002','X0001','X0002','X0001','X0002'],  'SCORE':[90,70,85,80,69,65,85,95]  })  # index dfGrade = STUDENT\_ID and COURSE\_ID  dfGrade.set\_index(['STUDENT\_ID','COURSE\_ID'],inplace=True)  def score(x):  if x >= 80:  return 'A'  elif x >= 70:  return 'B'  elif x >= 60:  return 'C'  elif x >= 50:  return 'D'  else:  return 'F'  # create function score for condition grade  dfGrade['GRADE'] = dfGrade['SCORE'].apply(score)  dfGrade |

### 3. Create a new DataFrame dfStudentGrade following information

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **STUDENT\_ID** | **DISPLAYNAME** | **GENDER** | **Database** | **Database\_Grade** | **Programming** | **Programming\_Grade** |
| 10001 | C. Evans | Male | 90 | A | 70 | B |
| 10002 | B. Larson | Female | 85 | A | 80 | A |
| 10003 | T. Holland | Male | 69 | C | 65 | C |
| 10004 | M. Ruffalo | Male | 85 | A | *NaN* | *NaN* |
| 10005 | S. Johansson | Female | *NaN* | *NaN* | 95 | A |

* GENDER: Transform data M = Male, F = Female
* Database: Database Score
* Database\_Grade: Database Grade
* Programming: Programming Score
* Programming\_Grade: Programming Grade

**Code:**

|  |
| --- |
| import pandas as pd  # create dfStudent  dfStudent = pd.DataFrame({  'STUDENT\_ID':['10001','10002','10003','10004','10005'],  'NAME':['Chris Evans','Brie Larson','Tom Holland','Mark Ruffalo','Scarlett Johansson'],  'GENDER':['M','F','M','M','F']},  )  # index dfStudent = STUDENT\_ID  dfStudent.set\_index('STUDENT\_ID',inplace=True)  # create dfCourse  dfCourse = pd.DataFrame({  'COURSE\_ID':['X0001','X0002'],  'NAME':['Database','Programming']  })  # index dfCourse = COURSE\_ID  dfCourse.set\_index('COURSE\_ID',inplace=True)  # create dfGrade  dfGrade = pd.DataFrame({  'STUDENT\_ID':['10001','10001','10002','10002','10003','10003','10004','10005'],  'COURSE\_ID':['X0001','X0002','X0001','X0002','X0001','X0002','X0001','X0002'],  'SCORE':[90,70,85,80,69,65,85,95]  })  # index dfGrade = STUDENT\_ID and COURSE\_ID  dfGrade.set\_index(['STUDENT\_ID','COURSE\_ID'],inplace=True)  # create columns FIRSTNAME,LASTNAME,DISPLAYNAME in dfStudent  dfStudent['FIRSTNAME'] = dfStudent['NAME'].apply(lambda x: x.split(' ')[0])  dfStudent['LASTNAME'] = dfStudent['NAME'].apply(lambda x: x.split(' ')[1])  dfStudent['DISPLAYNAME'] = dfStudent['NAME'].apply(lambda x: x[:1]) + '. ' + dfStudent['NAME'].apply(lambda x: x.split(' ')[1])  def score(x):  if x >= 80:  return 'A'  elif x >= 70:  return 'B'  elif x >= 60:  return 'C'  elif x >= 50:  return 'D'  else:  return 'F'  # create function score for condition grade  dfGrade['GRADE'] = dfGrade['SCORE'].apply(score)  dfGrade  # columes GENDER swap m and f to male and female  dfStudent['GENDER'].replace(['M','F'], ['MALE','FEMALE'],inplace = True)  dfStudent  # reset index  try:  dfStudent.reset\_index(level =['STUDENT\_ID'],inplace = True)  except:  pass  dfStudent  # create dfStudentGrade  dfStudentGrade = dfStudent[['STUDENT\_ID','DISPLAYNAME','GENDER']]  # reset index dfGrade  dfGrade.reset\_index(level =['COURSE\_ID','STUDENT\_ID'],inplace = True)  # create df\_XOO1 is dataframe course\_id = 'XOO1'  df\_XOO1 = dfGrade[dfGrade['COURSE\_ID'] == 'X0001']  # rename a columns score and grade to Database and Database\_Grade  df\_XOO1.rename(columns={'SCORE':'Database','GRADE':'Database\_Grade'},  inplace=True)  # create df\_XOO2 is dataframe course\_id = 'X0002'  df\_XOO2 = dfGrade[dfGrade['COURSE\_ID'] == 'X0002']  # rename a columns score and grade to Programming and Programming\_Grade  df\_XOO2.rename(columns={'SCORE':'Programming','GRADE':'Programming\_Grade'},  inplace=True)  # merge df\_XOO1 and df\_XOO2  df\_X\_ALL = pd.merge(df\_XOO1, df\_XOO2, how='outer', on=['STUDENT\_ID'])  df\_X\_ALL  # merge dfStudentGrade and df\_X\_ALL  dfStudentGrade = pd.merge(dfStudentGrade, df\_X\_ALL, how='left', on=['STUDENT\_ID'])  dfStudentGrade  # delete columns COURSE\_ID\_x and COURSE\_ID\_y  dfStudentGrade.drop(['COURSE\_ID\_x', 'COURSE\_ID\_y'], axis=1,inplace=True)  dfStudentGrade |

### 4. Create a new DataFrame dfAllStudentGrade following information

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **STUDENT\_ID** | **DISPLAYNAME** | **GENDER** | **COURSE\_NAME** | **SCORE** | **GRADE** | **SEQUENCE** |
| 10001 | C. Evans | Male | Database | 90 | A | 1 |
| 10002 | B. Larson | Female | Database | 85 | A | **2** |
| 10004 | M. Ruffalo | Male | Database | 85 | A | **2** |
| 10003 | T. Holland | Male | Database | 69 | C | 4 |
| 10005 | S. Johansson | Female | Database | *<NULL>* | I | 5 |
| 10005 | S. Johansson | Female | Programming | 95 | A | 1 |
| 10002 | B. Larson | Female | Programming | 80 | A | 2 |
| 10001 | C. Evans | Male | Programming | 70 | B | 3 |
| 10003 | T. Holland | Male | Programming | 65 | C | 4 |
| 10004 | M. Ruffalo | Male | Programming | *<NULL>* | I | 5 |

* SEQUENCE: Order data by score descending of each course. If more than one student have the same score, they should be in the same sequence (see example in sample data).

**Code:**

|  |
| --- |
| import pandas as pd  # create dfStudent  dfStudent = pd.DataFrame({  'STUDENT\_ID':['10001','10002','10003','10004','10005'],  'NAME':['Chris Evans','Brie Larson','Tom Holland','Mark Ruffalo','Scarlett Johansson'],  'GENDER':['M','F','M','M','F']},  )  # index dfStudent = STUDENT\_ID  dfStudent.set\_index('STUDENT\_ID',inplace=True)  # create dfCourse  dfCourse = pd.DataFrame({  'COURSE\_ID':['X0001','X0002'],  'NAME':['Database','Programming']  })  # index dfCourse = COURSE\_ID  dfCourse.set\_index('COURSE\_ID',inplace=True)  # create dfGrade  dfGrade = pd.DataFrame({  'STUDENT\_ID':['10001','10001','10002','10002','10003','10003','10004','10005'],  'COURSE\_ID':['X0001','X0002','X0001','X0002','X0001','X0002','X0001','X0002'],  'SCORE':[90,70,85,80,69,65,85,95]  })  # index dfGrade = STUDENT\_ID and COURSE\_ID  dfGrade.set\_index(['STUDENT\_ID','COURSE\_ID'],inplace=True)  # create columns FIRSTNAME,LASTNAME,DISPLAYNAME in dfStudent  dfStudent['FIRSTNAME'] = dfStudent['NAME'].apply(lambda x: x.split(' ')[0])  dfStudent['LASTNAME'] = dfStudent['NAME'].apply(lambda x: x.split(' ')[1])  dfStudent['DISPLAYNAME'] = dfStudent['NAME'].apply(lambda x: x[:1]) + '. ' + dfStudent['NAME'].apply(lambda x: x.split(' ')[1])  def score(x):  if x >= 80:  return 'A'  elif x >= 70:  return 'B'  elif x >= 60:  return 'C'  elif x >= 50:  return 'D'  else:  return 'F'  # create function score for condition grade  dfGrade['GRADE'] = dfGrade['SCORE'].apply(score)  dfGrade  # columes GENDER swap m and f to male and female  dfStudent['GENDER'].replace(['M','F'], ['MALE','FEMALE'],inplace = True)  dfStudent  # reset index  try:  dfStudent.reset\_index(level =['STUDENT\_ID'],inplace = True)  except:  pass  dfStudent  # create dfStudentGrade  dfStudentGrade = dfStudent[['STUDENT\_ID','DISPLAYNAME','GENDER']]  # reset index dfGrade  dfGrade.reset\_index(level =['COURSE\_ID','STUDENT\_ID'],inplace = True)  # create df\_XOO1 is dataframe course\_id = 'XOO1'  df\_XOO1 = dfGrade[dfGrade['COURSE\_ID'] == 'X0001']  # rename a columns score and grade to Database and Database\_Grade  df\_XOO1.rename(columns={'SCORE':'Database','GRADE':'Database\_Grade'},  inplace=True)  # create df\_XOO2 is dataframe course\_id = 'X0002'  df\_XOO2 = dfGrade[dfGrade['COURSE\_ID'] == 'X0002']  # rename a columns score and grade to Programming and Programming\_Grade  df\_XOO2.rename(columns={'SCORE':'Programming','GRADE':'Programming\_Grade'},  inplace=True)  # merge df\_XOO1 and df\_XOO2  df\_X\_ALL = pd.merge(df\_XOO1, df\_XOO2, how='outer', on=['STUDENT\_ID'])  df\_X\_ALL  # merge dfStudentGrade and df\_X\_ALL  dfStudentGrade = pd.merge(dfStudentGrade, df\_X\_ALL, how='left', on=['STUDENT\_ID'])  dfStudentGrade  # delete columns COURSE\_ID\_x and COURSE\_ID\_y  dfStudentGrade.drop(['COURSE\_ID\_x', 'COURSE\_ID\_y'], axis=1,inplace=True)  # course name in condition  def name\_course(id\_name):  if id\_name == 'X0001':  return 'Database'  else:  return 'Programming'  dfGrade['COURSE\_NAME'] = dfGrade['COURSE\_ID'].apply(name\_course)  # drop COURSE\_ID  dfGrade.drop('COURSE\_ID', axis=1,inplace=True)  # all dataframe dfgrade and df student  df\_all\_profile = pd.merge(dfGrade, dfStudent, how='outer', on=['STUDENT\_ID'])  df\_all\_profile.drop(['FIRSTNAME', 'LASTNAME','NAME'], axis=1,inplace=True)  # condition course\_name = database  df\_Database = df\_all\_profile[dfGrade['COURSE\_NAME'] == 'Database']  # sort column score in database  df\_Database.sort\_values(by=['SCORE'],ascending=False)  # all column sequence to sort rank value in database  df\_Database['SEQUENCE'] = df\_Database.groupby('COURSE\_NAME')['SCORE'].rank(ascending=False)  df\_Database.sort\_values(by=['SEQUENCE'], inplace=True)  # condition course\_name = Programming  df\_Programming = df\_all\_profile[dfGrade['COURSE\_NAME'] == 'Programming']  # all column sequence to sort rank value  df\_Programming.sort\_values(by=['SCORE'],ascending=False)  # all column sequence to sort rank value in programming  df\_Programming['SEQUENCE'] = df\_Programming.groupby('COURSE\_NAME')['SCORE'].rank(ascending=False)  df\_Programming.sort\_values(by=['SEQUENCE'], inplace=True)  df\_Programming  # concat to dfAllStudentGrade  dfAllStudentGrade = pd.concat([df\_Database,df\_Programming])  dfAllStudentGrade = dfAllStudentGrade.astype(int, errors='ignore')  dfAllStudentGrade |

## Problem 3: Create a method to find the result following conditions

Input: Distinct list of string e.g. { “A”, “B”, “C” }

Output: List of possible result set from input combination following the conditions:

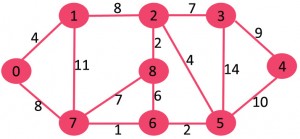
* No empty list e.g. { }
* No entire list e.g. { “A”, “B”, “C” }
* Number of members in list must be more than one item
* **Note:** You can calculate number of results by using the formula.
* Number of results = ; where n is number of elements in list

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| [“A”] | [] |
| [“A”,”B”] | [] |
| [ “A”, “B”, “C”] | [[“A”, “B”], [“A”, “C”], [“B”, “C”]] |
| [ “A”, “B”, “C”, “D”] | [[“A”,”B”], [“A”,”C”], [“A”,”D”], [“B”,”C”], [“B”,”D”], [“C”,”D”],  [“A”,“B”,”C”], [“A”,“B”,”D”],  [“A”,“C”,”D”], [“B”,“C”,”D”]] |
| [“A”,”B”,”C”,”D”,”E”,”F”] | [['A', 'B'], ['A', 'C'], ['A', 'D'], ['A', 'E'], ['A', 'F'], ['B', 'C'], ['B', 'D'], ['B', 'E'], ['B', 'F'], ['C', 'D'], ['C', 'E'], ['C', 'F'], ['D', 'E'], ['D', 'F'], ['E', 'F'], ['A', 'B', 'C'], ['A', 'B', 'D'], ['A', 'B', 'E'], ['A', 'B', 'F'], ['A', 'C',  'D'], ['A', 'C', 'E'], ['A', 'C', 'F'], ['A', 'D', 'E'], ['A', 'D', 'F'], ['A', 'E', 'F'], ['B', 'C', 'D'], ['B', 'C', 'E'], ['B', 'C', 'F'], ['B', 'D', 'E'], ['B', 'D', 'F'], ['B', 'E', 'F'], ['C', 'D', 'E'], ['C', 'D', 'F'], ['C', 'E', 'F'], ['D', 'E', 'F'], ['A', 'B', 'C', 'D'], ['A', 'B', 'C', 'E'], ['A', 'B', 'C', 'F'], ['A', 'B', 'D', 'E'], ['A', 'B', 'D', 'F'], ['A', 'B', 'E', 'F'], ['A', 'C', 'D', 'E'], ['A', 'C', 'D', 'F'], ['A', 'C', 'E', 'F'], ['A', 'D', 'E', 'F'], ['B', 'C', 'D', 'E'], ['B', 'C', 'D', 'F'], ['B', 'C', 'E', 'F'], ['B', 'D', 'E', 'F'], ['C', 'D', 'E', 'F'], ['A', 'B', 'C', 'D', 'E'], ['A',  'B', 'C', 'D', 'F'], ['A', 'B', 'C', 'E', 'F'], ['A', 'B', 'D', 'E', 'F'], ['A', 'C', 'D', 'E', 'F'], ['B', 'C', 'D', 'E', 'F']] |

**Method:**

|  |
| --- |
| from itertools import combinations  def FindPossibleList(inputList):  # all posible list  list\_all = sum([list(map(list, combinations(inputList, i))) for i in range(len(inputList) + 1)], [])  # all answer list  list\_answer = list\_all[len(inputList) + 1:-1]  return list\_answer  print(FindPossibleList(['A', 'B', 'C','D','E','F'])) |

## Problem 4: Finding shortest path from Node 0 to Node 4

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/Fig-0.jpg)Input:

|  |  |  |
| --- | --- | --- |
| **Src** | **Dest** | **Weight** |
| 0 | 1 | 4 |
| 0 | 7 | 8 |
| 1 | 2 | 8 |
| 1 | 7 | 11 |
| 2 | 3 | 7 |
| 2 | 5 | 4 |
| 3 | 4 | 9 |
| 3 | 5 | 14 |
| 5 | 4 | 10 |
| 6 | 5 | 2 |
| 7 | 6 | 1 |
| 7 | 8 | 7 |
| 8 | 2 | 2 |
| 8 | 6 | 6 |

Output: shortest path from node 0 to node 4

**Code:**

|  |
| --- |
| # variable declaration graph node  graph = {  "0":{"1":4,"7":8},  "1":{"2":8,"7":11},  "2":{"1":8,"3":7,"5":4,"8":2},  "3":{"2":7,"4":9,"5":14},  "4":{"3":9,"5":10},  "5":{"2":4,"3":14,"4":10},  "6":{"5":2,"7":1,"8":6},  "7":{"0":8,"6":1,"8":7},  "8":{"2":2,"6":6,"7":7}  }  def dijkstra(graph,start,goal):  shortest\_distance = {}  predecessor = {}  unseenNodes = graph  infinity = 9999999  path = []  # enter node in loop  for node in unseenNodes:  shortest\_distance[node] = infinity  # start = 0  shortest\_distance[start] = 0  # enter node in loop  while unseenNodes:  minNode = None  for node in unseenNodes:  if minNode is None:  # way is near  minNode = node  elif shortest\_distance[node] < shortest\_distance[minNode]:  minNode = node  # all way in loop  for childNode, weight in graph[minNode].items():  if weight + shortest\_distance[minNode] < shortest\_distance[childNode]:  shortest\_distance[childNode] = weight + shortest\_distance[minNode]  predecessor[childNode] = minNode  unseenNodes.pop(minNode)  # node goal  currentNode = goal  while currentNode != start:  try:  path.insert(0,currentNode)  currentNode = predecessor[currentNode]  except KeyError:  print('Path not reachable')  break  path.insert(0,start)  if shortest\_distance[goal] != infinity:  print('Shortest distance is ' + str(shortest\_distance[goal]))  print('And the path is ' + str(path))  dijkstra(graph, "0", "4")  # Shortest distance is 21  # And the path is ['0', '7', '6', '5', '4'] |