2-more_exercises

September 6, 2024

1 Python review: More exercises

This notebook continues the review of Python basics. A key concept is that of a *nested* data structure. For example, the first code cell will define a 2-D "array" as a list of lists.

```
In [42]: %load_ext autoreload
         %autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Consider the following dataset of exam grades, organized as a 2-D table and stored in Python as a "list of lists" under the variable name, `grades`.

```
In [43]: grades = [
             # First line is descriptive header. Subsequent lines hold data
             ['Student', 'Exam 1', 'Exam 2', 'Exam 3'],
             ['Thorny', '100', '90', '80'],
             ['Mac', '88', '99', '111'],
             ['Farva', '45', '56', '67'],
             ['Rabbit', '59', '61', '67'],
             ['Ursula', '73', '79', '83'],
             ['Foster', '89', '97', '101']
         ]

         grades
```

```
Out[43]: [['Student', 'Exam 1', 'Exam 2', 'Exam 3'],
          ['Thorny', '100', '90', '80'],
          ['Mac', '88', '99', '111'],
          ['Farva', '45', '56', '67'],
          ['Rabbit', '59', '61', '67'],
          ['Ursula', '73', '79', '83'],
          ['Foster', '89', '97', '101']]
```

Exercise 0 (`students_test`: 1 point). Complete the function `get_students` which takes a nested list `grades` as a parameter and reutrns a new list, `students`, which holds the names of the students as they from "top to bottom" in the table. - **Note**: the parameter `grades` will be similar to the table above in structure, but the data will be different.

```
In [44]: def get_students(grades):
             ###
             ### YOUR CODE HERE
             ###
             students = []
             for record in grades[1:]:
                 students.append(record[0])
             return students
```

The demo cell below should display `['Thorny', 'Mac', 'Farva', 'Rabbit', 'Ursula', 'Foster']`.

```
In [45]: students = get_students(grades)
         students

Out[45]: ['Thorny', 'Mac', 'Farva', 'Rabbit', 'Ursula', 'Foster']
```

The test cell below will check your solution against several randomly generated test cases. If your solution does not pass the test (or if you're just curious), you can look at the variables used in the latest test run. They are automatically imported for you as part of the test.

- `input_vars` - Dictionary containing all of the inputs to your function. Keys are the parameter names.
- `original_input_vars` - Dictionary containing a copy of all the inputs to your function. This is useful for debugging failures related to your solution modifying the input. Keys are the parameter names.
- `returned_output_vars` - Dictionary containing the outputs your function generated. If there are multiple outputs, the keys will match the names mentioned in the exercise instructions.
- `true_output_vars` - Dictionary containing the outputs your function **should have** generated. If there are multiple outputs, the keys will match the names mentioned in the exercise instructions.

All of the test cells in this notebook will use the same format, and you can expect a similar format on your exams as well.

```
In [46]: # `students_test`: Test cell
         import nb_1_2_tester
         tester = nb_1_2_tester.Tester_1_2_0()
         for _ in range(20):
             try:
                 tester.run_test(get_students)
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
             except:
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
                 raise
         print('Passed. Please submit!')

Passed. Please submit!
```

Exercise 1 (`assignments_test`: 1 point). Complete the function `get_assignments`. The function takes `grades` (a nested list structured similarly to `grades` above) as a parameter. It should return a new list `assignments` which holds the names of the class assignments. (These appear in the descriptive header element of `grades`.)

```
In [47]: def get_assignments(grades):
             ###
             ### YOUR CODE HERE
             ###
             header = grades [0]
             assignments = header[1:]
             return assignments
```

The demo cell below should display `['Exam 1', 'Exam 2', 'Exam 3']`

```
In [48]: assignments = get_assignments(grades)
         assignments

Out[48]: ['Exam 1', 'Exam 2', 'Exam 3']
```

The following test cell will check your function against several randomly generated test cases.

```
In [49]: import nb_1_2_tester
         tester = nb_1_2_tester.Tester_1_2_1()
         for _ in range(20):
             try:
                 tester.run_test(get_assignments)
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = t
             except:
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = t
                 raise
         print('Passed. Please submit!')

Passed. Please submit!
```

Exercise 2 (`grade_lists_test`: 1 point). Complete the function for `build_grade_lists`, again taking `grades` as a parameter. The function should return a new *dictionary*, named `grade_lists`, that maps names of students to *lists* of their exam grades. The grades should be converted from strings to integers. For instance, `grade_lists['Thorny'] == [100, 90, 80]`.

```
In [50]: # Create a dict mapping names to lists of grades.
         def build_grade_lists(grades):
             ###
             ### YOUR CODE HERE
             ###
             grade_lists = {}
             for record in grades [1:]:
                 student_name = record[0]
                 student_grades = [int(grade) for grade in record[1:]]
                 grade_lists[student_name] = student_grades
             return grade_lists
```

The demo cell below should display

```
{'Thorny': [100, 90, 80],
 'Mac': [88, 99, 111],
 'Farva': [45, 56, 67],
 'Rabbit': [59, 61, 67],
 'Ursula': [73, 79, 83],
 'Foster': [89, 97, 101]}
```

```
In [51]: grade_lists = build_grade_lists(grades)
         grade_lists

Out[51]: {'Thorny': [100, 90, 80],
          'Mac': [88, 99, 111],
          'Farva': [45, 56, 67],
          'Rabbit': [59, 61, 67],
          'Ursula': [73, 79, 83],
          'Foster': [89, 97, 101]}
```

```
In [52]: import nb_1_2_tester
         tester = nb_1_2_tester.Tester_1_2_2()
         for _ in range(20):
             try:
                 tester.run_test(build_grade_lists)
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
             except:
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
                 raise
         print('Passed. Please submit!')

Passed. Please submit!
```

Exercise 3 (`grade_dicts_test`: 2 points). Complete the function `build_grade_dicts`, again taking `grades` as a parameter and returning new dictionary, `grade_dicts`, that maps names of students to *dictionaries* containing their scores. Each entry of this scores dictionary should be keyed on assignment name and hold the corresponding grade as an integer. For instance, `grade_dicts['Thorny']['Exam 1'] == 100`. You may find solutions to earlier exercises useful for completing this one. Feel free to use them!

```
In [53]: def build_grade_dicts(grades):
             ###
             ### YOUR CODE HERE
             ###
             assignments = grades[0][1:]
             grade_dicts = {}
             for record in grades[1:]:
                 student_name = record[0]
                 student_scores = {assignments[i]: int(record[i+1]) for i in range(len(assignme
                 grade_dicts[student_name] = student_scores
             return grade_dicts
```

The demo cell below should display {'Thorny': {'Exam 1': 100, 'Exam 2': 90, 'Exam 3': 80}, 'Mac': {'Exam 1': 88, 'Exam 2': 99, 'Exam 3': 111}, 'Farva': {'Exam 1': 45, 'Exam 2': 56, 'Exam 3': 67}, 'Rabbit': {'Exam 1': 59, 'Exam 2': 61, 'Exam 3': 67}, 'Ursula': {'Exam 1': 73, 'Exam 2': 79, 'Exam 3': 83}, 'Foster': {'Exam 1': 89, 'Exam 2': 97, 'Exam 3': 101}}

```
In [54]: grade_dicts = build_grade_dicts(grades)
         grade_dicts
```

```
Out[54]: {'Thorny': {'Exam 1': 100, 'Exam 2': 90, 'Exam 3': 80},
          'Mac': {'Exam 1': 88, 'Exam 2': 99, 'Exam 3': 111},
          'Farva': {'Exam 1': 45, 'Exam 2': 56, 'Exam 3': 67},
          'Rabbit': {'Exam 1': 59, 'Exam 2': 61, 'Exam 3': 67},
          'Ursula': {'Exam 1': 73, 'Exam 2': 79, 'Exam 3': 83},
          'Foster': {'Exam 1': 89, 'Exam 2': 97, 'Exam 3': 101}}
```

```
In [55]: import nb_1_2_tester
         tester = nb_1_2_tester.Tester_1_2_3()
         for _ in range(20):
             try:
                 tester.run_test(build_grade_dicts)
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
             except:
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
                 raise
         print('Passed. Please submit!')
```

```
Passed. Please submit!
```

Exercise 4 (avg_grades_by_student_test: 1 point). Complete the function build_avg_by_student, taking grades as a parameter and returning a dictionary named avg_by_student that maps each student to his or her average exam score. For instance, avg_grades_by_student['Thorny'] == 90. You may find solutions to earlier exercises useful for completing this one. Feel free to use them!

Hint. The statistics module of Python has at least one helpful function.

```
In [56]: # Create a dict mapping names to grade averages.
         def build_avg_by_student(grades):
             ###
             ### YOUR CODE HERE
             ###
             avg_by_student = {}
             grade_lists = build_grade_lists(grades)
             for student, grades_list in grade_lists.items():
                 avg = sum(grades_list) / len(grades_list)
                 avg_by_student[student] = avg
             return avg_by_student
```

The demo cell below should display

```
{'Thorny': 90,
 'Mac': 99.33333333333333,
 'Farva': 56,
 'Rabbit': 62.333333333333336,
 'Ursula': 78.33333333333333,
 'Foster': 95.66666666666667}
```

```
In [57]: avg_grades_by_student = build_avg_by_student(grades)
         avg_grades_by_student
```

```
Out[57]: {'Thorny': 90.0,
          'Mac': 99.33333333333333,
          'Farva': 56.0,
          'Rabbit': 62.333333333333336,
          'Ursula': 78.33333333333333,
          'Foster': 95.66666666666667}
```

```
In [58]: import nb_1_2_tester
         tester = nb_1_2_tester.Tester_1_2_4()
         for _ in range(20):
             try:
                 tester.run_test(build_avg_by_student)
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = t
             except:
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = t
                 raise
         print('Passed. Please submit!')
```

```
Passed. Please submit!
```

Exercise 5 (grades_by_assignment_test: 2 points). Complete the function build_grade_by_asn, which takes grades as a parameter and returns a dictionary named grade_by_asn, whose keys are assignment (exam) names and whose values are lists of scores over all students on that assignment. For instance, `grades_by_assignment['Exam 1'] == [100, 88, 45, 59, 73, 89]`. You may find solutions to earlier exercises useful for completing this one. Feel free to use them!

```
In [59]: def build_grade_by_asn(grades):
             ###
             ### YOUR CODE HERE
             ###
             assignments = grades[0][1:]
             grade_by_asn = {assignment: [] for assignment in assignments}
             for record in grades[1:]:
                 for i, assignment in enumerate(assignments):
                     grade_by_asn[assignment].append(int(record[i+1]))
             return grade_by_asn
```

The demo cell below should display

```
{'Exam 1': [100, 88, 45, 59, 73, 89],
 'Exam 2': [90, 99, 56, 61, 79, 97],
 'Exam 3': [80, 111, 67, 67, 83, 101]}
```

```
In [60]: grades_by_assignment = build_grade_by_asn(grades)
         grades_by_assignment
```

```
Out[60]: {'Exam 1': [100, 88, 45, 59, 73, 89],
          'Exam 2': [90, 99, 56, 61, 79, 97],
          'Exam 3': [80, 111, 67, 67, 83, 101]}
```

```
In [61]: import nb_1_2_tester
         tester = nb_1_2_tester.Tester_1_2_5()
         for _ in range(20):
             try:
                 tester.run_test(build_grade_by_asn)
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
             except:
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
                 raise
         print('Passed. Please submit!')
```

```
Passed. Please submit!
```

Exercise 6 (avg_grades_by_assignment_test: 1 point). Complete the function build_avg_by_asn, which takes grades as a parameter and returns a dictionary, avg_by_asn, which maps each exam to its average score. You may find solutions to earlier exercises useful for completing this one. Feel free to use them!

```
In [64]: # Create a dict mapping items to average for that item across all students.

         def build_avg_by_asn(grades):

             grade_by_asn = build_grade_by_asn(grades)

             avg_by_asn = {}

             for assignment, grades_list in grade_by_asn.items():
                 avg = sum(grades_list) / len(grades_list)
                 avg_by_asn[assignment] = avg
             return avg_by_asn
```

The demo cell below should display

```
{'Exam 1': 75.66666666666667,
 'Exam 2': 80.33333333333333,
 'Exam 3': 84.83333333333333}
```

```
In [65]: avg_grades_by_assignment = build_avg_by_asn(grades)
         avg_grades_by_assignment

Out[65]: {'Exam 1': 75.66666666666667,
          'Exam 2': 80.33333333333333,
          'Exam 3': 84.83333333333333}

In [66]: import nb_1_2_tester
         tester = nb_1_2_tester.Tester_1_2_6()
         for _ in range(20):
             try:
                 tester.run_test(build_avg_by_asn)
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
             except:
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
                 raise
         print('Passed. Please submit!')

Passed. Please submit!
```

Exercise 7 (`rank_test`: 2 points). Complete the function `get_ranked_students` which takes `grades` as an argument and returns a new list, `ranked_students`, which contains the names of students in order by *decreasing* score. That is, `ranked_students[0]` should contain the name of the top student (highest average exam score), and `ranked_students[-1]` should have the name of the bottom student (lowest average exam score). You may find solutions to earlier exercises useful for completing this one. Feel free to use them!

The demo cell below shuould display `['Mac', 'Foster', 'Thorny', 'Ursula', 'Rabbit', 'Farva']`

```
In [71]: def get_ranked_students(grades):
             ###
             ### YOUR CODE HERE
             ###
             avg_by_student = build_avg_by_student(grades)
             ranked_students = sorted(avg_by_student, key=avg_by_student.get, reverse = True)

             return ranked_students

In [72]: rank = get_ranked_students(grades)
         rank

Out[72]: ['Mac', 'Foster', 'Thorny', 'Ursula', 'Rabbit', 'Farva']

In [73]: import nb_1_2_tester
         tester = nb_1_2_tester.Tester_1_2_7()
         for i in range(5):
             try:
                 tester.run_test(get_ranked_students)
```

```
                (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
            except:
                (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
                raise
        print('Passed. Please submit!')
```

```
Passed. Please submit!
```

Fin! You've reached the end of this part. Don't forget to restart and run all cells again to make sure it's all working when run in sequence; and make sure your work passes the submission process. Good luck!