# Python review: More exercises

This notebook continues the review of Python basics based on Chris Simpkins's Python Bootcamp.

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 [ ]:
```
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']
]
import numpy as np

Newgrade = []
a = np.array(grades)
Newgrade.append(list(a[0:,1]))
Newgrade.append(list(a[0:,2]))
Newgrade.append(list(a[0:,3]))
```

In [ ]:
```
a
```

**Exercise 0** (`students_test`: 1 point). Write some code that computes a new list named `students[:]`, which holds the names of the students as they from "top to bottom" in the table.

In [ ]:
```
#
# YOUR CODE HERE
#
```

In [ ]:
```
# `students_test`: Test cell
print(students)
assert type(students) is list
assert students == ['Thorny', 'Mac', 'Farva', 'Rabbit', 'Ursula', 'Foster']
print("\n(Passed!)")
```

**Exercise 1** (`assignments_test`: 1 point). Write some code to compute a new list named `assignments[:]`, to hold the names of the class assignments. (These appear in the descriptive header element of `grades`.)

In [ ]:
```
#
# YOUR CODE HERE
#
```

In [ ]:
```
# `assignments_test`: Test cell
print(assignments)
assert type(assignments) is list
assert assignments == ['Exam 1', 'Exam 2', 'Exam 3']
print("\n(Passed!)")
```

**Exercise 2** (`grade_lists_test`: 1 point). Write some code to compute 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 [ ]:
```
# Create a dict mapping names to lists of grades.
#
# YOUR CODE HERE
#
```

In [ ]:
```
# `grade_lists_test`: Test cell
print(grade_lists)
assert type(grade_lists) is dict, "Did not create a dictionary."
assert len(grade_lists) == len(grades)-1, "Dictionary has the wrong number of entries."
assert {'Thorny', 'Mac', 'Farva', 'Rabbit', 'Ursula', 'Foster'} == set(grade_lists.keys()), "Dictionary has the wrong keys."
assert grade_lists['Thorny'] == [100, 90, 80], 'Wrong grades for: Thorny'
assert grade_lists['Mac'] == [88, 99, 111], 'Wrong grades for: Mac'
assert grade_lists['Farva'] == [45, 56, 67], 'Wrong grades for: Farva'
assert grade_lists['Rabbit'] == [59, 61, 67], 'Wrong grades for: Rabbit'
assert grade_lists['Ursula'] == [73, 79, 83], 'Wrong grades for: Ursula'
assert grade_lists['Foster'] == [89, 97, 101], 'Wrong grades for: Foster'
print("\n(Passed!)")
```

**Exercise 3** (`grade_dicts_test`: 2 points). Write some code to compute a 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`.

In [ ]:
```
grade_lists.values()
```

In [ ]:
```
grades_working = grades[:]
student_copy = students[:]
#assignments is the variable from a previous question.

#grade_lists is from problem 1.2.2

grades_working = grades[:]
student_copy = students[:]

grade_dicts = {}
for i,val in grade_lists.items():
     grade_dicts[i] = dict(zip(assignments,val))
print(grade_dicts)
```

In [ ]:
```
# Create a dict mapping names to dictionaries of grades.
#
# YOUR CODE HERE
#
```

In [ ]:
```
# `grade_dicts_test`: Test cell
print(grade_dicts)
assert type(grade_dicts) is dict, "Did not create a dictionary."
assert len(grade_dicts) == len(grades)-1, "Dictionary has the wrong number of entries."
assert {'Thorny', 'Mac', 'Farva', 'Rabbit', 'Ursula', 'Foster'} == set(grade_dicts.keys()), "Dictionary has the wrong keys."
assert grade_dicts['Foster']['Exam 1'] == 89, 'Wrong score'
assert grade_dicts['Foster']['Exam 3'] == 101, 'Wrong score'
assert grade_dicts['Foster']['Exam 2'] == 97, 'Wrong score'
assert grade_dicts['Ursula']['Exam 1'] == 73, 'Wrong score'
assert grade_dicts['Ursula']['Exam 3'] == 83, 'Wrong score'
assert grade_dicts['Ursula']['Exam 2'] == 79, 'Wrong score'
assert grade_dicts['Rabbit']['Exam 1'] == 59, 'Wrong score'
assert grade_dicts['Rabbit']['Exam 3'] == 67, 'Wrong score'
assert grade_dicts['Rabbit']['Exam 2'] == 61, 'Wrong score'
assert grade_dicts['Mac']['Exam 1'] == 88, 'Wrong score'
assert grade_dicts['Mac']['Exam 3'] == 111, 'Wrong score'
assert grade_dicts['Mac']['Exam 2'] == 99, 'Wrong score'
assert grade_dicts['Farva']['Exam 1'] == 45, 'Wrong score'
assert grade_dicts['Farva']['Exam 3'] == 67, 'Wrong score'
assert grade_dicts['Farva']['Exam 2'] == 56, 'Wrong score'
assert grade_dicts['Thorny']['Exam 1'] == 100, 'Wrong score'
assert grade_dicts['Thorny']['Exam 3'] == 80, 'Wrong score'
assert grade_dicts['Thorny']['Exam 2'] == 90, 'Wrong score'
print("\n(Passed!)")
```

**Exercise 4** (`avg_grades_by_student_test`: 1 point). Write some code to compute a dictionary named `avg_grades_by_student` that maps each student to his or her average exam score. For instance, `avg_grades_by_student['Thorny'] == 90`.

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

In [ ]:
```
# Create a dict mapping names to grade averages.
#
# YOUR CODE HERE
#
```

In [ ]:
```
# `avg_grades_by_student_test`: Test cell
print(avg_grades_by_student)
assert type(avg_grades_by_student) is dict, "Did not create a dictionary."
assert len(avg_grades_by_student) == len(students), "Output has the wrong number of students."
assert abs(avg_grades_by_student['Mac'] - 99.33333333333333) <= 4e-15, 'Mean is incorrect'
assert abs(avg_grades_by_student['Foster'] - 95.66666666666667) <= 4e-15, 'Mean is incorrect'
assert abs(avg_grades_by_student['Farva'] - 56) <= 4e-15, 'Mean is incorrect'
assert abs(avg_grades_by_student['Rabbit'] - 62.333333333333336) <= 4e-15, 'Mean is incorrect'
assert abs(avg_grades_by_student['Thorny'] - 90) <= 4e-15, 'Mean is incorrect'
assert abs(avg_grades_by_student['Ursula'] - 78.33333333333333) <= 4e-15, 'Mean is incorrect'
print("\n(Passed!)")
```

**Exercise 5** (`grades_by_assignment_test`: 2 points). Write some code to compute a dictionary named `grades_by_assignment`, 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]`.

In [ ]:
```
grades
```

In [ ]:
```
#
# YOUR CODE HERE
#
```

In [ ]:
```
# `grades_by_assignment_test`: Test cell
print(grades_by_assignment)
assert type(grades_by_assignment) is dict, "Output is not a dictionary."
assert len(grades_by_assignment) == 3, "Wrong number of assignments."
assert grades_by_assignment['Exam 1'] == [100, 88, 45, 59, 73, 89], 'Wrong grades list'
assert grades_by_assignment['Exam 3'] == [80, 111, 67, 67, 83, 101], 'Wrong grades list'
assert grades_by_assignment['Exam 2'] == [90, 99, 56, 61, 79, 97], 'Wrong grades list'
print("\n(Passed!)")
```

**Exercise 6** (`avg_grades_by_assignment_test`: 1 point). Write some code to compute a dictionary, `avg_grades_by_assignment`, which maps each exam to its average score.

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

In [ ]:
```
# `avg_grades_by_assignment_test`: Test cell
print(avg_grades_by_assignment)
assert type(avg_grades_by_assignment) is dict
assert len(avg_grades_by_assignment) == 3
assert abs((100+88+45+59+73+89)/6 - avg_grades_by_assignment['Exam 1']) <= 7e-15
assert abs((80+111+67+67+83+101)/6 - avg_grades_by_assignment['Exam 3']) <= 7e-15
assert abs((90+99+56+61+79+97)/6 - avg_grades_by_assignment['Exam 2']) <= 7e-15
print("\n(Passed!)")
```

**Exercise 7** (`rank_test`: 2 points). Write some code to create a new list, `rank`, which contains the names of students in order by *decreasing* score. That is, `rank[0]` should contain the name of the top student (highest average exam score), and `rank[-1]` should have the name of the bottom student (lowest average exam score).

In [ ]:
```
#
# YOUR CODE HERE
#
```

In [ ]:
```
# `rank_test`: Test cell
print(rank)
print("\n=== Ranking ===")
for i, s in enumerate(rank):
    print("{}. {}: {}".format(i+1, s, avg_grades_by_student[s]))

assert rank == ['Mac', 'Foster', 'Thorny', 'Ursula', 'Rabbit', 'Farva']
for i in range(len(rank)-1):
    assert avg_grades_by_student[rank[i]] >= avg_grades_by_student[rank[i+1]]
print("\n(Passed!)")
```

In [ ]:
```
sorted(temp_people, key=temp_people.get, reverse=True)
```

In [ ]:
```
sorted_grades=(list(avg_grades_by_student.values()))
sorted_grades.sort(reverse=True)
temp_people ={'Mac': sorted_grades[0], 'Foster': sorted_grades[1], 'Thorny' : sorted_grades[2],
              'Ursula': sorted_grades[3], 'Rabbit':sorted_grades[4], 'Farva':sorted_grades[4]}
rank = [v for v in sorted(temp_people, key=temp_people.get, reverse=True)]
print(sorted_grades, rank)
```

**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!