Revision History

Version 01/07/2020

1. Corrected the test case output for `average_exams`
2. Corrected the test case output for `calc_final_grades`
3. Made minor changes to the test cases for `date_cruncher`
4. Corrected the parameters for `date_cruncher` in the template file
5. Added `pass` under each function in the template file

Important

1. Due Date: **01/15/2020 at 11:59 pm**
2. This homework is graded out of **100** points. **The GradeScope autograder will only show up to 90 points when you submit your code. We are saving some test cases for the final autograder that we will use.** This is to encourage you to start developing ways to test features of your code when you aren't given test cases in an autograder. We also reserve the right to change the test cases later on as we see fit. Therefore, the grade reflected in GradeScope does not reflect your final grade on HW01.py
3. This is an individual assignment. You may collaborate with other students in this class. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. Students may only collaborate with fellow students currently taking CS 2316, the TA's and the lecturer. You should not exchange code or write code for others. For individual assignments, each student must turn in a unique program. Your submission must not be substantially similar to another student's submission. Collaboration at a reasonable level will not result in substantially similar code.
4. For Help:
   - TA Helpdesk (Schedule posted on class website)
   - Email TA's or use Piazza Forums Notes
   - How to Think Like a Computer Scientist http://openbookproject.net/thinkcs/python/english3e/
5. Test your code first by including your own function calls, then comment out or delete your function calls before submitting to GradeScope. GradeScope will not work if it has any code not within one of the required functions.
6. Do not wait until the last minute to do this assignment in case you run into problems.
7. Read the entire specifications document before starting this assignment.

Introduction

The goal of this homework is to showcase your knowledge of basic python, namely variables, expressions, functions, conditionals, loops, strings, tuples, lists, and dictionaries. You should test your functions out first on your own computer. Then when you have one or more functions working, comment out your test cases and print statements and upload the entire file (which must be named **HW01.py**) to GradeScope to see how well your code performs on the test cases we have created. You can submit the homework file as many times as you'd like before the deadline.

Background

In this assignment, there will not be a common central theme, instead we have the functions categorized by the Python topics covered.

Make sure you write your code in the template we provided to you (HW01.py).

Functions for HW01.py

Note: See the grading rubric at the end of the document for point distribution.

Expressions and Conditionals

Function name: **`split_bill`**
Parameters: `num_people` (int), `tip_percent` (float), `pizza_cost` (int), `tax_percent` (float)
Returns: float
Description: You and a few friends are having a meal at a pizza restaurant, and the server has just given you the bill. Write a function that calculates the total cost of the meal including tax and gratuity, and returns how much each person must pay rounded to 2 decimal places. Note that the tax should be calculated and added to the total, and the tip amount should be determined based on the total including tax. Additionally, you may assume that the parameters will always follow the types listed above. The percentages given for tax and tip will always be between 0 and 1 inclusive. There will always be at least one person.
Test Cases:

```
>>> split_bill(2, .25, 15, 0.07)
10.03
>>> split_bill(4, 0.0, 39, 0.25)
12.19
```

Iteration, Lists, and Tuples

Function name: **`contains`**
Parameters: `a_list` (list), `value` (int/float/str)
Return Type: tuple
Description:
This function should iterate through `a_list` using a for loop and return a boolean indicating if the value is within `a_list` and also return the number of iterations necessary to find the element. Your loop must not execute more iterations than necessary, and you **cannot** use the **in** operator. Further you should iterate through `a_list` sequentially and should not manipulate the order or values of the list object. The boolean and number of iterations should be returned as a tuple.
Test Cases:

```
>>> contains([1, 2, “3”, 4.5], 3)
(False, 4)
>>> contains([1, 2, “3”, 4.5], 4.5)
(True, 4)
```

Function name: **score**
Parameters: `a_tuple` (tuple)
Return Type: int
Description: This function should calculate the score of a given football team when `a_tuple` is formatted such that the zeroth element is the number of touchdowns, the first element is the number of extra points, the second element is the number of two point conversions, the third element is the number of field goals, and the fourth element is the number of safeties. To calculate the score, you should sum the number of touchdowns multiplied by 6, the number of extra points, the number of two point conversions multiplied by 2, the number of field goals multiplied by 3, and the number of safeties multiplied by 2. You may assume `a_tuple` has exactly 5 elements.
Test Case:

```
>>> score((2, 2, 0, 1, 0))
17
```

Dictionaries

Function name: **winner**
Parameters: `a_dict` (dict)
Return Type: str
Description: This function should return the name of the winning football team when `a_dict` has the following format… `{ “Team1” : (1, 1, 0, 0, 0), “Team2” : (1, 1, 0, 1, 0) }`. You may assume that there are only two teams and that there will not be a tie, i.e. both scores will never be the same. Hint: It may be helpful to call the *score* function you wrote previously.
Test Case:

```
>>> a_dict = {
               "Texans" : (2, 0, 2, 2, 0),
               "Bills" : (1, 1, 0, 4, 0)
            }
>>> winner(a_dict)
“Texans”
```

For the next two functions assume the parameter `super_grades` is a list of lists with the following structure and types …

```
super_grades = [
                 [ 'Student', 'Exam 1', 'Exam 2', 'Exam 3', 'HW Avg'],
                 [    'Eric',    '100',     '90',     '80',     '90'],
                 [    'Josh',     '88',     '99',    '111',     '92'],
                 [   'Manny',     '45',     '56',     '67',     '54'],
                 [ 'Colleen',     '59',     '61',     '67',     '72'],
                 [  'Bohong',     '73',     '79',     '83',     '85'],
                 [ 'Marylyn',     '89',     '97',    '101',     '80'],
                 [ 'Xinran',      '40',     '50',    '100',     '87']
                ]
```

Function name: **`average_exams`**
Parameters: `super_grades` (list)
Return Type: dict
Description: This function should return a dictionary that maps the exam title to the average score for that exam rounded to two decimal places. You should not hard code the keys to the dictionary, but you may assume that the format of `super_grades` will stay the same. You may further assume that there will be exactly three exams.
Test Case:

```
>>> average_exams(super_grades)
{
'Exam 1': 70.57,
'Exam 2': 76.0,
'Exam 3': 87.0
}
```

Function name: **`calc_final_grades`**
Parameters: `super_grades` (list)
Return Type: dict
Description: This function should return a dictionary that maps the students name to their final letter grade. The final grade is calculated by taking `0.25 * ( <Exam 1> + <Exam 2> + <Exam 3> + <HW Avg> )`. When determining the letter grade, use the same grade cut offs defined in the syllabus.
Test Case:

```
>>> calc_final_grades(super_grades)
{
 'Eric': 'A',
 'Josh': 'A',
 'Manny': 'F',
 'Colleen': 'D',
 'Bohong': 'B',
 'Marylyn': 'A',
 'Xinran': 'D'
}
```

Strings

Function name: **`date_cruncher`**
Parameters: `original_dates` (list of strings)
Return Type: list
Description: Write a function that takes in a list of dates in “{month name} {date}, {year}” format, formats them into “{MM}-{DD}-{YYYY}” format, and returns the list of formatted dates. In `original_dates`, the month could be the full month name OR a three-letter abbreviation of the month name (e.g., “October” vs. “Oct”). You should account for the month name having any type of capitalization. The date could be just a number (e.g., “18”, “30”) or the ordinal number (e.g., “1st”, “2nd”, “3rd”, “4th”, etc.). You can assume the year will be 4 digits long. The formatted dates in the list returned should have the month and date both appear as two-digit integers.
Test Cases:

```
>>> original_dates = ["March 2nd, 2019", "July 8th, 1997",
"September 19th, 4001", "January 1st, 0001"]
>>> date_cruncher(original_dates)
['03-02-2019', '07-08-1997', '09-19-4001', '01-01-0001']

>>> original_dates = ["dEcEmBER 3, 2020", "FEB 31st, 6000", "apR
5, 1776"]
>>> date_cruncher(original_dates)
['12-03-2020', '02-31-6000', '04-05-1776']
```

Testing Your Code

Use function calls and print statements to verify that your code performs correctly on the test cases provided as well as your own test cases. Be sure to comment out or delete any function calls or print statements before submitting to GradeScope.

GradeScope Requirements

- NO UNNECCESARY PRINT STATEMENTS.
- NO UNNECCESARY FUNCTION CALLS
- Make sure the file submitted is titled HW01.py
- Only submit HW01.py file

Grading Rubric

split_bill:	10 pts
contains:	10 pts
score:	10 pts
winner:	15 pts
average_exams:	15 pts
calc_final_grades:	20 pts
date_cruncher:	20 pts
Total	100 pts