Instructions

In a Python file, a comment starts with # and extends to the end of the line, e.g.

This is a comment.

Comments are ignored by the Python processor.

1. In the IDLE editor, create a new file **`hw1.py`**
2. At the top of the file, enter

```
# cmps5P
# hw1
# <your student id>
```

3. Enter the solutions to the problems below in file **`hw1.py`**.
   Precede each problem with a comment stating the problem number.
   For instance, start your answer to problem 1) a) with

```
# 1a
```

4. Register and submit to Crowdgrader as follows:

   Self-enroll here:
   http://www.crowdgrader.org/crowdgrader/venues/join/1731/bysowi_sudyga_pemenu_vemiqu

   Then upload your file hw1.py to Crowdgrader.org.

5. Due date:
   submission: end of day on 25 Jan 2016
   peer reviews: end of day on 29 Jan 2016

Problems

1) [10 points] Defining and calling math functions

a) [2] Define a function **`average(a,b)`** that computes the average of **a** and **b**; i.e.

$$average(a, b) = \frac{a + b}{2}$$

Show the results for the test set [(7,3), (6.5, 2.5)]; i.e.
i.e. add two comment lines after the function definition of the form

```
# average(7,3) == <result>
# average(6.5, 3.5) == <result>
```

b) [2] Define a function **`square(a)`** that squares its argument; i.e.

$$square(a) = a^2$$

Show the results for the test set [2, 2.5].

c) [2] Define a function **`sum_squares(a,b)`** that computes the sum of the squares of its arguments; i.e.

$$sum\_squares(a, b) = a^2 + b^2$$

Show the results for the test set [(7,3), (6.5, 3.5)].

d) [2] Define a function **`square_sum(a,b)`** that computes the sum of the square of the sum of its arguments; i.e.

$$square\_sum(a, b) = (a + b)^2$$

Show the results for the test set [(7,3), (6.5, 3.5)].

e) [2] Define a function **`f(a,b)`** that computes the following formula:

$$f(a, b) = \frac{(a + b)^2}{4} - \frac{a^2 + b^2}{2}$$

Can you define function **f** using only the functions **`square`**, **`sum_squares`**, **`and square_sum`**, and the operation minus?
Note: You may transform the formula into an algebraically equivalent form.

Show the results for the test set [(7,3), (6.5, 3.5)].

2) [15 points] A little moving help

a) [5] You need to move a bunch of similar sized items, e.g. books. Boxes are available in two sizes, large and small. A large box can hold 20 items; a small box holds 7 items. Define a function **`boxes207()`** that given a number of items will compute how many boxes of each size are needed so that the total number of boxes is kept to a minimum.

Define **`boxes207`** so that it will ask to input the number of items:

```
>>> boxes207()
How many items do you need to move? 39
```

The answer should look like this:

```
You will need 1 large box(es) and 3 small box(es).
```

b) [10] Define a function **`boxes(large, small)`** that improves on **`boxes207`** in two ways:

i) The improved function is parameterized on the large and small box size.
ii) It uses the correct singular and plural form with the number of boxes needed.

Here are some sample calls to **`boxes`**:

```
>>> boxes(20, 7)
How many items do you need to move? 39
Answer:
You will need 1 large box and 3 small boxes.

>>> boxes(20, 7)
How many items do you need to move? 6
Answer:
You will need 0 large boxes and 1 small box.

>>> boxes(15, 6)
How many items do you need to move? 39
Answer:
You will need 2 large boxes and 2 small boxes.
```

Hints:

a) Consider using the operations // (integer division) and % (modulo).
b) The Python function `int(s)` converts a numeral ("12") into an int value; `str(i)` turns integer 12 into string "12".
c) Review the Python constructs for conditional statements and conditional expressions.
d) It may help to review the files luggage.py and my_math.py (posted on Piazza).

3) [15] Taking and filling orders at the restaurant Tutto Italiano

Your favorite Italian restaurant Tutto Italiano offers spaghetti, pizza, gnocchi, and lasagna. The tables are numbered from 1 to 20.

We will write a small application to support the service in Tutto Italiano.

We model an order as a dictionary
`{'table': <table number>, 'dish': <name of dish>}`
Table numbers are Python int values (from 1 to 20). Dishes are represented by their names as Python strings ('pizza').

a) [5] We keep track of the list of open orders with a global variable **`orders`**;
initially the list **`orders`** is empty.

The function **`take_order(ord)`** appends the given order **`ord`** to the list of **`orders`**:
`orders.append(ord)` adds **`ord`** as the last element to the list of **`orders`**.

The function **`fill_order()`** removes the first element from the list **`orders`**:
`ord = orders.pop(0)`
removes the first order from the list of **`orders`** and stores it in **`ord`**.

Furthermore, let's say the current order **`ord == {'table': 9, 'dish': 'spaghetti'}.`**
Then **`fill_order()`** will print **`ord`** in the form:
`Order for table 9: spaghetti`

Test your system by making the following function calls in IDLE (after (re)loading module hw1):
```
take_order({'table':  7, 'dish': 'lasagna'})
take_order({'table': 11, 'dish': 'gnocchi'})
fill_order()
```
After these calls, access the value of orders and record it as a comment after function **`fill_order`**.

b) [5] Let's upgrade our ordering system for Tutto Italiano by adding some accounting:
Define a global variable **`sales`** which keeps track of how many orders of each dish have been sold; initialize
`sales = {'gnocchi': 0, 'lasagna': 0, 'pizza': 0, 'spaghetti': 0}`

Every time an order is received, we need to increase the count for the dish ordered.
Thus, define a function **`take_order_acc(ord)`** that
(1) like **`take_order()`** adds the order to the list of open orders and
(2) increases the count for the dish ordered in sales.
For instance,
`sales['lasagna'] += 1`
increases the lasagna count by 1.

c) [3] Define a function **`init_Tutto_italiano()`** that (re)sets the variables **`orders`** and **`sales`** to their initial values.

d) [2] Test your system by making the following function calls in IDLE (after (re)loading module hw1):

```
init_Tutto_Italiano()
take_order({'table':  7, 'dish': 'lasagna'})
take_order({'table': 11, 'dish': 'gnocchi'})
fill_order()
```

After these calls, access the value of **`orders`** and **`sales`** and record them as comments after function **`fill_order_acc`**.