```
# - ProblemSet1.py *- coding: utf-8 -*-
"""
Each problem will be a function to write.

Remember that you can execute just the code between the #%% signs by clicking
somewhere in that space and then using Ctrl-Enter (Cmd-Return on a Mac). An
alternative is to use the second toolbar green triangle or Menu>Run>Run cell.

On loops especially, you can make an error that causes the program to run
forever. If you don't get immediate response, then this is probably happening.
In that case, try Ctrl-C. If that doesn't stop it, click your IPython console
away and open a new one from the Consoles menu. Look over your code and see
why the termination condition can't be met and fix it. Then run again.

You can submit your work in two ways. One is simply to upload this file. This
is the easiest way to go provided that you haven't corrupted it.  To make sure
that it isn't corrupted, run the whole file by clicking the left green triangle
above.  If it is corrupted, you can fix it or submit just your function as
described next.
To submit only a single function, copy the material between the two #%% into
a text file using this Spyder editor (Choose menu File>New File... to get a
new file and then copy your function from between the #%%'s. Don't use a word
processor, but you can use a text editor such as Notepad in Windows or TextEdit
on the Mac. Save your new file as problemx.py (where x is the problem number).
Upload this file to Coursera.  Note that the grading program is going to run
the function with the name specified, so don't change the function name (it
doesn't matter what you name the file itself).

Note: each of the functions below is made runnable by adding the statement
pass.  This is a do-nothing statement.  You should replace it with your code,
but its present doesn't affect how your code runs.
"""
"""
IMPORTANT TIPS:
1) When you upload your problem(s) you click on SUBMIT at the bottom of the
page.  That will start the auto-grader.  It should return with your grade
within a few minutes.

2) You can resubmit any problem as often as you would like and find out how
you did in minutes. This is the advantage of the autograder over peer grading
used in many courses. With peer grading you'd have to grade several other
students' problems at the end of the week, you wouldn't find out how you did
for at least a week, and you couldn't resubmit after seeing your result.

3) The downside of the auto-grader is that it is very strict and literal. You
should format your output EXACTLY as shown in the example runs.

4) Common mistakes: including an extra space in the printout, mis-spelling a
word, changing the punctuation that the example uses, and using end=" " when
not needed.

5) The example run uses example data.  The auto-grader will always use
different data, so don't write functions that are specific to the example data.
In particular, don't use the variable name of the example data within your
function --- it will likely be completely undefined and unknown to the grader.

6) These are test questions that can be resubmitted.  So the auto-grader gives
fairly generic error messages -- it does not tell you how to fix your function.
It will say that your code produced too few lines or too many lines; it will
say whether your code failed x number of test cases and give you a fractional
score out of 10 rounded to the nearest integer. For example, if the function
should output 3 lines and you got 2 of the 3 right, it would say failed one
case and give you a score of 7 (2/3 of 10 rounded)).  Sometimes, we run your
function several times on different data.  In this case it will tell you the
```

```
number of cases (i.e., runs) that failed and give you a fractional score. It
may tell you that your function failed to run. Often if your program crashes,
there will be lots of error messages produced and the grader will tell you it
produced too many lines.

7) Run your program on the example data and make sure it matches the example
output.  If you program doesn't get full credit when graded, check the format
of the print statements first.

8) All of the exercises in this course can be solved using the material in the
course lectures.  There is no need to look to outside sources of more material.
In fact, a sophisticated outside technique can sometimes have side-effects that
you aren't aware of that causes your function to fail the grader.

"""

"""
Problem 1_1:
Write a function problem1_1() that prints "Problem Set 1".

Tip: Be careful that your program outputs this exact phrase with no additional
characters.  It will be graded automatically and must be precise.  Here is the
run of my solved problem 1_1:

problem1_1()
Problem Set 1

Note the problem1_1() is what I typed to run the problem and "Problem Set 1" is
what it printed out.  There will typically be a sample run such as this either
before or after the statement of each problem. This helps clarify what you
are expected to do and shows how the auto-grader expects it to look.

"""
#%%
def problem1_1():
    pass # replace this pass (a do-nothing) statement with your code


#%%

"""
Problem 1_2:
Write a function problem1_2(x,y) that prints the sum and product of the
numbers x and y on separate lines, the sum printing first.
"""
#%%
def problem1_2(x,y):
    pass # replace this pass (a do-nothing) statement with your code


#%%
"""
Test run. Note that the grader program will use different numbers:

problem1_2(3,5)
8
15
"""
"""
Problem 1_3:
Write a function problem1_3(n) that adds up the numbers 1 through n and
prints out the result. You should use either a 'while' loop or a 'for' loop.
Be sure that you check your answer on several numbers n.  Be careful that your
loop steps through all the numbers from 1 through and including n.
```

```
Tip: As this involves a loop you could make an error that causes it to run
forever. Usually Control-C will stop it. See suggestions at the beginning of
this document.  With loops take care that your first and last iterations are
what you expect. A print statement can be inserted in the loop to monitor it,
but be sure this isn't in the submitted function.
"""
#%%
def problem1_3(n):
    my_sum = 0
    pass # replace this pass (a do-nothing) statement with your code





#%%
"""
Test run. Note that the grader program will use a different number for n:

problem1_3(6)
21
"""
"""
Problem 1_4:
Write a function 'problem1_4(miles)' to convert miles to feet. There are
5280 feet in each mile. Make the print out a statement as follows:
"There are 10560 feet in 2 miles."  Except for the numbers this statement
should be exactly as written.

Tip: Watch the spacing before and after your numbers.  Make sure that it is
just one space or the auto-grader may not give you credit.
"""
#%%
def problem1_4(miles):
    pass # replace this pass (a do-nothing) statement with your code



#%%
"""
Test run. Note that the grader program will use different numbers:

problem1_4(5)
There are 26400 feet in 5 miles.
"""
"""
Problem 1_5:
Write a function 'problem1_5(age)'. This function should use if-elif-else
statement to print out "Have a glass of milk." for anyone under 7; "Have
a coke." for anyone under 21, and "Have a martini." for anyone 21 or older.

Tip: Be careful about the ages 7 (a seven year old is not under 7) and 21.
Also be careful to make the phrases exactly as shown for the auto-grader.
"""
#%%
def problem1_5(age):
    pass # replace this pass (a do-nothing) statement with your code





#%%
"""
Test runs (3 of them). Note that the grader program will use different numbers:
```

```
problem1_5(5)
Have a glass of milk.

problem1_5(10)
Have a coke.

problem1_5(25)
Have a martini.

"""
"""
Problem 1_6:
Write a function 'problem1_6()' that prints the odd numbers from 1 through 100.
Make all of these numbers appear on the same line (actually, when the line
fills up it will wrap around, but ignore that.). In order to do this, your
print statement should have end=" " in it. For example, print(name,end=" ")
will keep the next print statement from starting a new line. Be sure there is a
space between these quotes or your numbers will run together. Use a single
space as that is what the grading program expects. Use a 'for' loop
and a range() function.

Things to be careful of that might go wrong: You print too many numbers, you
put too much or too little space between them, you print each number on its
own line, you print even numbers or all numbers, your first number isn't 1 or
your last number isn't 99.  Always check first and last outputs when you write
a loop.
"""
#%%
def problem1_6():
    pass # replace this pass (a do-nothing) statement with your code




#%%
"""
Test run (I've inserted a newline here to cause wrapping in the editor):

problem1_6()
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55
57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
"""
"""
Problem 1_7:
Write a function problem1_7() that computes the area of a trapezoid. Here is the
formula: A = (1/2)(b1+b2)h. In the formula b1 is the length of one of the
bases, b2 the other. The height is h and the area is A. Basically, this
takes the average of the two bases times the height. For a rectangle b1 = b2,
so this reduces to b1*h. This means that you can do a pretty good test of the
correctness of your function using a rectangle (that way you can compute the
answer in your head). Use input statements to ask for the bases and the height.
Convert these input strings to real numbers using float(). Print the output
nicely EXACTLY like mine below.

Tip: Be careful that your output on the test case below is exactly as shown
so that the auto-grader judges your output correctly.  The auto-grader does
not look at your input statements, so you don't have to use my input prompts
if you don't want to. However, the auto-grader will enter the three inputs in
the order shown. See the other test run below.

problem1_7()

Enter the length of one of the bases: 3

Enter the length of the other base: 4
Enter the height: 8
The area of a trapezoid with bases 3.0 and 4.0 and height 8.0 is 28.0

"""
#%%
def problem1_7():
    pass # replace this pass (a do-nothing) statement with your code







#%%
"""
Another test run. In grading, expect different input numbers to be used.

problem1_7()

Enter the length of one of the bases: 10

Enter the length of the other base: 11

Enter the height: 12
The area of a trapezoid with bases 10.0 and 11.0 and height 12.0 is 126.0

"""
```