Instructions

These exercises are designed to help cement the concepts from lecture and give you an opportunity to apply them yourself. You are welcome to discuss and share ideas with each other, but should write the answers and code yourself.
Some questions ask you to complete a checkoff, which means you need to come to office hours at this link and talk to a staff member. Staff members may complete checkoffs for multiple students at one time, meaning you may not answer every question yourself but instead should listen to what your peers have to say and ask questions! Collaboration is an important part of computer science.

1 Tasks

1.1 print vs return - print_vs_return.py

Open the template file **print_vs_return.py**. In it, these two functions are defined:

```
# print_vs_return.py
def f1(x):
    print(x + 1)

def f2(x):
    return x + 1
```

Run this code. We will now go through some examples using these functions. Answer all the bold questions by commenting in **print_vs_return.py**.

What happens when we call the functions below? (Try getting the answer on your own and use IDLE (interactively) only to check your answer).

Example 1:

```
# Example 1:
f1(3)
# ?

f2(3)
# ?
```

From example 1, it looks like they behave in exactly the same way, but they really don't.

Example 2: Try this:

```
# Example 2:
a = f1(3)
b = f2(3)
a
# ?
b
# ?
```

What are the values of a and b? Why are they different?

Example 3: Now run the following code:

```
# Example 3:
print(f1(3))
# ?
print(f2(3))
# ?
```

What is the output? Why are you getting this output?

Example 4:

```
# Example 4:
f1(3) + 1
# ?
f2(3) + 1
# ?
```

What is the output? Why are you getting this output?

Note: You might find it useful to explicitly **return None** when you don't want your function to return anything. This way, you'll always remember that what Python is actually doing is returning **None** if you don't tell it to return anything.

> *Checkoff #1—Explain your results to a course staff! What do all of these examples tell us about when we want to return and when we want to print? (When is printing useful? When is returning useful?)*

1.2 Mystery program - mystery.py

Do not run this program. Just look at it and determine outputs. Look at this code fragment and trace what's happening. Using a pen and paper to walk through the code might be helpful! Find out what the final outcome is and show your work by creating a file called **mystery.py** and commenting in it.

```
def b(z):
    prod = a(z, z)
    print(z, prod)
    return prod

def a(x, y):
    x = x + 1
    prod = x * y
    return prod

def c(x, y, z):
    total = x + y + z
    power = b(total) ** 2
    return power

x = 1
y = x + 1
res = c(x, y+3, x+y)
print(res)
```

> *Checkoff #2—Once you are done, you can run the code to verify that your answer is correct. Did anything surprise you? What functions were called in which order?*

1.3 Not equal - not_equal.py

Imagine that the **!=** operator for numbers has suddenly been removed from the Python programming language. We need to find a way to replace it with the remaining operators in the language. Remember that before it went missing, the **!=** operator took two numbers, and returned **True** if they were not equal to one another, and **False** otherwise.

Create a file called **not_equal.py** and write a function **not_equal**. We still have the **not** operator (which can be applied to booleans) and the **==** operator, which can be applied to numbers.

Now let's try and test our new function. Create 4 variables **a**, **b**, **c**, and **d**, such that **a** and **b** have the same value, and **c** and **d** are different from one another.

Now write a call to your **not_equal** function, passing it **a** and **b**, and assign its return value to a variable **not_equal_test1**. Then print out the value of **a** and **b** and the value that was returned by your function **not_equal**. This tests whether your function does the right thing when the two values are actually equal.

Now test that your function works when the parameters are not equal (by passing it **c** and **d**). Assign the return value to a variable **not_equal_test2**, and print out the values.

Compare your output to the output below: (Note: Your output will be slightly different if you choose different values for **a**, **b**, **c**, and **d**.)

```
# sample output where a = 5, b = 5, c = 1, d = 3
5
5
False
1
3
True
```

> *Checkoff #3—Can you think of any variables a,b,c,d that would return the wrong result with your implementation on accident?*

1.4 Making desserts with Python - desserts.py

Use the provided code to "make" desserts with Python.

> **Hint**: Remember the differences between **return** and **print**. Do we want our functions to return or print in this problem?

(a) We've given you a very basic recipe for tiramisu – you combine ladyfingers, mascarpone cream, and espresso syrup. However, it just comes out in one big chunk. We need you to be organized chefs and make a separate function for each of the parts.

Make separate functions for ladyfingers, mascarpone cream, and espresso syrup. The functions should print out the individual recipes for each of those parts. Make a function for tiramisu that calls each of the functions of its component parts. When you run it, it should output the same thing as the original program did, but be much more organized in the code!

(b) Now you should make a new function that prints out a nice border, something like this:

_____———-**********ooooooOOOOOooooo**********———-_____

or this:

¡¿¡¿¡¿¡¿¡¿¡¿¡¿[][][][][][][][]¡¿¡¿¡¿¡¿¡¿¡¿¡¿¡¿¡¿

Print the border out, using the function you just created, at the beginning of the recipe, after the title, and at the end.

(c) You're a very innovative chef and have come up with a new recipe – ladyfinger parfaits. This recipe is one part ladyfingers, which you already have a recipe for, and one part parfait cream.

Copy the parfait cream recipe from **parfaitcream.txt** and paste it into **desserts.py**. (You can highlight the text with your mouse, then copy and paste.) Then edit your code to make a new function that prints out the recipe, like you did for the ladyfingers recipe above. Then make a new function for your combined ladyfinger parfaits recipe, too – using the ladyfinger function and parfait cream function.

Print all of the recipes now, with your border before the titles, after the titles, and after the text of the recipe.

2 Optional Challenge[1]:

2.1 Yummy graphics - parfait_graphics.py

We already know how to print text in the Python shell, but sometimes we may want to "draw" text, so that we can decide its color, size, and more. The code in **parfait_graphics.py** is an example of how to draw text on the screen. Run the program to see the text it draws in the window.

Now, change the font size, style, and color of the text. (You can pick whatever you like!) Look at the **setSize**, **setStyle**, and **setTextColor** methods in the documentation under Text methods. All the **set** methods that change the attributes of the graphics object automatically update its appearance on the screen.

After you've played around, use the functions you wrote in Problem 4 in **desserts.py** to "draw" your recipe for ladyfinger parfaits on the screen. (Hint: Change the **print** statements to draw text on the screen instead. You will need to choose the locations to "draw" each line of text!)

Create a function called **parfait_stamp** that takes in an **x** and a **y** value and draws a ladyfinger parfait centered around the point (x, y) on the screen. (Don't worry: we're not grading for realism.) Decorate your recipe!

Submitting your PSET

After you've finished your PSET and checkoff: Log into your Canvas account, find the post for Problem Set 5 in Assignments, and submit all of the files that you created or edited. After you turn in your assignment, you're all done!

[1] optional problems are provided for those who have further interests and want to explore more. You are not responsible for those questions, however.