MTH 3300
E. Fink

3300 Problems, Section 6: Functions

1. Consider the following program:

```
def f(a):
   print(a)
   a = a + 1
   return a + 2

def main():
   x = 2
   y = f(x)
   print(x, y)

main()
```

   What will print out?

2. Determine what the following code displays.

```
def fn(x, y):
   x = x + 1
   y = y + 2
   return x + y

def main():
   a = 10
   y = 20
   c = fn(y,a)
   d = fn(c,a)
   print(a,y,c,d)

main()
```

3. Determine what the following code displays.

```
def add1_to_first(x):
   x[0] += 1
   return x

def main():
   y = [4,5,6]
   z = add1_to_first(y)
   print(y, z)

main()
```

4. What will be displayed by the following?

```
glob1 = 5
glob2 = 6

def fn(x):
   x += glob1
   glob2 = x
   return [x, glob2]

def main():
   print(fn(3), glob1, glob2)

main()
```

5. What will print when the following code is run?

```
def fn(my_num, my_list):
   my_num += 1
   my_list[0] += 1


a_number = 10
a_list = [4,8,2,3]
x = fn(a_number, a_list)
print(a_number)
print(a_list)
print(x)
```

6. Write a function called `update`, which takes one list parameter as input, adds 1 to each entry, and returns *nothing.* For example, when the following code is run, the list `[5, 7, 9, 10]` should print out:

```
x = [4, 6, 8, 9]
update(x)
print(x)
```

7. Recall the formula $e^x \approx 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}$, where n can be any integer greater than 1. The higher the value of n, the more accurate your estimate is.

   Write a program that asks the user to input a value of x, and a power n, that outputs an approximation to e^x using the terms up to power n. For example, if the user enter `0.1` for x and `3` for n, the program should output `1.10517`, because $1 + \frac{0.1}{1!} + \frac{0.1^2}{2!} + \frac{0.1^3}{3!} = 1.10517$ (when rounded).

   For full credit, accomplish this using two functions. One should be called `myExp` that computes the approximation: this function should have two parameters, a `float` named `x` and an `int` named `n`, and should return the approximate value of e^x using the terms up to that power. The other should be called `fact`: it should take an `int n` as input, and return `n` factorial.

8. A **pyramidal** number is a number that is the sum of the first n consecutive square integers. So, the first pyramidal number is 1 (since $1^2 = 1$); the second pyramidal number is 5 (since $1^2 + 2^2 = 5$); the third pyramidal number is 14 (since $1^2 + 2^2 + 3^2 = 14$); and so on.

   Write a program that allows the user to input an integer n, and the prints out the **first n pyramidal numbers.** For example, if the user enters `3`, the program would print out the first 3 pyramidal numbers: `1`, `5`, and `14`, each on a different line.

   For full credit, write and use a **function** called `pyr` whose input is an integer `x`, and whose output is the `x`th pyramidal number. For example, the value of `pyr(3)` should be 14.

   Your program and function only need to work if the inputs are positive integers.

9. Write a program that asks the user to enter integer values n and k, and then computes n choose k (you may have seen this written as ${}_nC_k$ or $\left(\begin{array}{c} n \\ k \end{array}\right)$), which is given by the formula $\frac{n!}{k!(n-k)!}$. For example, ${}_5C_2 = \frac{5!}{2!3!} = \frac{120}{2 \cdot 6} = 10$.

   For full credit, use a **function** as part of your program – you probably want to write the factorial operation as a function; otherwise, you will need several loops, instead of just one!

   Your program only needs to work if the entered n and k values are 0 or positive.

10. Write a *whole* program that does the following: The program should create 8-letter passwords, made up of 8 random letters, until the user is satisfied.

    Specifically, it should do the following repeatedly: first, it should print out a random password. Then, it should ask the user if the password is satisfactory. If the user enters `y`, the program ends; if the user enters `n`, everything repeats. (You may assume the user enters only `y` or `n`.)

    For full credit, you should use a FUNCTION that takes NO arguments, and returns a `string` (the random password).

    Hints: to create a random letter, import `random` and `string`, and use `random.choice(string.ascii_letters)`. To create the full random password, start with an empty `string`, and `+=` random lowercase letters on to the end, one letter at a time.

11. Two words are called *granagrams* if they have the same number of g's AND the same number of r's in them. For instance, greening and reigning are granagrams, because they each have two g's and one r. ggrrrh and rrxkabgrg *are* granagrams, while rage and garbage are *not*.

    Write a *whole* program that does the following: first, it asks the user to enter two words. Then, the program will print Granagrams or Not Granagrams depending on whether those two words are granagrams. ASSUME that all letters entered are LOWERCASE, and each word contains no spaces.

    For full credit, you should use a FUNCTION – or perhaps a couple, your choice – which returns a count of the number of appearances of a certain letter in a string. So your function(s) should have at least one argument which is a string, and should return an int – beyond that is up to you. DO NOT USE .count()!

12. A *palindrome* is a word that reads the same forwards and backwards, for example racecar or abba. Write a program that checks whether an integer entered by the user is a palindrome. Example run:

    ```
    Enter a word:  racecar
    That's a palindrome!
    ```

    or:

    ```
    Enter a word:  plant
    Nope, not a palindrome.
    ```

    (In these examples, racecar and plant are user input.)

    You must do it as follows for full credit: first, have the user enter the word. Then, you should write a function called my_reverse, which takes in a string as an argument, and then returns a new string which is the input reversed. (So, for example, if x = "Hello", then when I write print(my_reverse(x)) it would print out olleH.) You should also use this function!

    Hints: In the function, you should start with a new empty string for output, and then loop through the input backwards, adding on one character to the input at a time. DO NOT USE reversed().

13. a. Write a function called all_ones, which receives a *non-empty* list of ints as an argument. The function should return True if all the entries are equal to 1, and False otherwise.

    b. I have a list of integers called lights, which starts out initialized with 10 values, each of which is a 0 or 1:

    ```
    lights = [0, 0, 1, 1, 0, 1, 0, 0, 1, 0]
    ```

    Write code that repeatedly picks one of the entries at random, and "switches" it: if the entry is a 0, it should be replaced with a 1, and if the entry is a 1, it should be replaced with a 0. The program should keep doing this until all the entries are 1's. Finally, once this has been achieved, the program should print out the *number of switches that have been made* (i.e., the number of times your loop has run). **For full credit, use the function you wrote in part a.**

14. a. Write a function called all_here which receives two lists of strings as arguments, and returns True if every element in the first list appears at least once in the second list (and False otherwise). For example, all_here(["a","b","c"], ["b","c","q","a","t","a"]) would return True since "a","b" and "c" all appear in the second list; whereas all_here(["a","b","c"], ["a","z","b"]) would return False since "c" does not appear in the second list.

    (Hint: to make this shorter, remember that you can check the presence of an element x in a list y by using the code if x in y.)

    b. I have a list of strings called participants:

    ```
    participants = ["Alice", "Bob", "Christine", "Dennis", "Evan", "Frank", "George", "Howard"]
    ```

    Write code that repeatedly asks the user to enter in a name, until they have entered in all the names in participants. Once this has happened, the program should print out the *total number of names that were entered.* **For full credit, use the function you wrote in part a.**