Problem 1
- What is the output of print(str[2:5]) if str = ‘Hello World!’?

  llo

- What is the output of print(str * 2) if str = ‘Hello World!’?

  Hello World!Hello World!

- What is the output of print(tuple * 2) if tuple = (123, ‘john’)?

  (123, ‘john’, 123, ‘john’)

Problem 2
- Write a function last_two(str) that will take a string called str as a parameter. The function will return the last two characters of the string. If the string has less than two characters, return an empty string.

  Example: last_two(“Hello World”) will return “ld”

def last_two(str):
  if(len(str) < 2):
    return ""
  else:
    return str[-1] + str[-2]

Another Solution:

def last_two(str):
  if(len(str) < 2):
    return ""
  return str[len(str):]
Problem 3
(For the following three mini-problems, we’d like you to get familiar with reading Python 3 documentation, so have someone in your team ready to look up string functions to help you solve these. https://docs.python.org/3/library/stdtypes.html#string-methods or Google Python 3 strings and go to the methods section)

- Write a function dash_list(str) that will take a string called str as a parameter. This string will be composed of words separated with dashes (the '-' character). The function will return a list of strings where each string in the list is one of the words in the original string. The list should not have any dashes in it.

Example: dash_list("CS-120-Summer-2017-U-of-A")
Should return ["CS", "120", "Summer", "2017", "U", "of", "A"]

```python
def dash_list(str):
    return str.split('-')
```

- Write a function comma_str(L) that will take a list of strings called L as a parameter. This function will return a string with commas inserted between the words. (Do not use a loop, use a string function)

Example: comma_str(["CS","120","Summer","2017","U","of","A"])
Should return “CS,120,Summer,2017,U,of,A”

```python
def comma_str(L):
    return ",".join(L)
```

- Write a function strip_case(str) that will take a string called str as a parameter. This function will return an uppercase version of the string with all exclamation points and question marks from both sides removed.

Example: strip_case("?!?!?!?!?!!??!how are you?!?!?!?!?!")
Should return “HOW ARE YOU”

```python
def strip_case(str):
    return str.strip('?!').upper()
```
Problem 4
- Write a function `sum_list(L)` that will take a list called `L` as a parameter. The function will loop over the list of numbers and add them. It should then print the sum and return it.
(Practicing loops is important, so do this with a for loop and then with a while loop. But is there a standard library function to make this faster? Have one of your team Google for an answer.)

Example: `sum_list([5,6,7])` should print “Sum 18” and return 18

```python
def sum_list(L):
    total = 0
    for num in L:
        total += num
    print("Sum " + str(total))
    return total
```

Problem 5
- Write Python code to read from two files, `testOutput.txt` and `realOutput.txt`, and compare each line in the file with the corresponding line in the other. If the lines are the same do nothing. If they are different print a message that says that there are differences between the files.

This solution assumes that both the files have the same number of lines, can you think of a solution that doesn't have this assumption?

```python
test = open("testOutput.txt").readlines()
real = open("realOutput.txt").readlines()

for i in range(len(test)):  # Assuming both files have the same length, you could do len(real) instead
    if(test[i] != real[i]):
        print("The lines are different")
```

Russ's solution:
```python
if open("testOutput.txt").readlines() != open("realOutput.txt").readlines():
    print("The lines are different")
```
Problem 6
- Write a function flip_dictionary(D) that will take a dictionary called D with integers as keys and strings as values as a parameter. The function will return a new dictionary where strings are keys and integers are values.

Example: flip_dictionary({1:"a", 2:"b", 3:"c", 4:"d", 5:"e", 6:"f"})
will return: { "a":1,"b":2,"c":3,"d":4,"e":5,"f":6}

def flip_dictionary(D):
    new_d = {}
    for key in D.keys():
        new_d[D[key]] = key
    return new_d

Bonus: Normally, this function returns a dictionary with the same number of items as its input. But occasionally, the new dictionary has fewer. Can you figure out why this might happen?

If two different keys have the same value, then when you flip the dictionary, then that value becomes a key. Dictionaries can only have unique keys, so the value that appears twice is gone. For example, the following dictionary:
x = {1:'a',2:'a',3:'b'}

A call to flip_dictionary() with x would return:
{ 'a':1,'b':3} or { 'a':2,'b':3}
Depending on which key-value pair gets inserted last. Shrinking the dictionary in both ways.