Python Programming Exercises 4

Notes: In the previous set of exercises we learnt how to use lists and for loops in our programs. In these exercises we will use another fundamental compound data type: the dictionary as well as another way to iterate, the while loop.

1. Dictionaries are declared as a list of comma separated key/value pairs between curly braces. Key and value are separated by a colon. An empty dictionary is created with just a pair of curly braces. You can use `len(···)` to get the number of key/value pairs in a dictionary:

```
>>> len({})
>>> len({ "key1" : "value1", "key2" : "value2" })
```

2. Keys are associated with values in the dictionary by indexing the dictionary with a key and assigning a value:

```
>>> d = {}
>>> d["key1"] = "value1"
>>> d["key2"] = "value2"
>>> d
```

3. We retrieve the value associated with a specific key by indexing the dictionary with the same key:

```
>>> d["key1"]
>>> d["key2"]
```

4. Dictionary keys must be immutable data types. For example, lists do not work, but strings, ints and floats do:

```
>>> d = {}
>>> d["key"] = 1
>>> d[17] = 0.125
>>> d[0.0] = "value"
>>> d
```

5. The inclusion operator tests whether keys exist or not:

```
>>> d = { "key1" : "value1", "key2" : "value2" }
>>> "key1" in d
>>> "key3" in d
```

6. Key/value pairs can be removed from a dictionary by using the **del** keyword:

```
>>> d = { "key1" : "value1", "key2" : "value2" }
>>> del d["key1"]
>>> d
```

Write a function called `remove_keys(`$mydict$`,`$keylist$`)` that accepts two parameters: a dictionary called $mydict$ and a list called $keylist$. `remove_keys(`$mydict$`,`$keylist$`)` should remove all the keys contained in $keylist$ from $mydict$ and return the dictionary:

```
d = { "key1" : "value1", "key2" : "value2", "key3" : "value3", "key4" : "value4" }
keys = ["key1", "key3", "key5"]

if remove_keys(d, keys) == { "key2" : "value2", "key4" : "value4" } :
    print("correct!")
```

7. When we iterate through a dictionary using a for loop, we actually iterate over the keys:

```
d = { "key1":1, "key2":2, "key3":1, "key4":3, "key5":1, "key6":4, "key7":2 }

for k in d :
    print("key=", k, " value=", d[k], sep="")
```

Modify the code above to print just the keys associated to values that are greater than 1.

8. Write a function called `accept_login`($users, username, password$) with three parameters: $users$ a dictionary of username keys and password values, $username$ a string for a login name and $password$ a string for a password. The function should return **True** if the user exists and the password is correct and **False** otherwise. Here is the calling code, test your code with both good and bad passwords as well as non-existent login names:

```
users = {
    "user1" : "password1",
    "user2" : "password2",
    "user3" : "password3"
}

if accept_login(users, "wronguser", "wrongpassword") :
    print("login successful!")
else :
    print("login failed...")
```

9. A **while** loop keeps iterating as long as a condition evaluates to **True**:

```
count = 0

while count < 5 :
    print("count =", count)
    count += 1
```

Use a while loop to print a triangle of astericks, like this:

```
    *
   ***
  *****
 *******
*********
```

10. Write a function that prints a triangle of astericks like the one in the previous question. The function should accept a single parameter, $height$, that defines how tall the triangle is (in the previous example $height = 5$). Use a while loop and ensure your function works by trying different heights.

11. In the previous set of exercises we rewrote the `sum()` function using a for loop (question 17). Reimplement it using a **while loop** instead.

12. When the condition for the while loop requires a lot of code, it is sometimes more readable to loop forever and explicitly use the **break** keyword. Fix the following code to do this:

```
attempts = 0

while True :
    response = input("Do you want to quit? (y/n): ")
    attempts += 1

print("Exiting after", attempts, "attempts")
```

13. Write a function called `find_value`($mydict, val$) that accepts a dictionary called $mydict$ and a variable of any type called val. The function should return a list of keys that map to the value val in $mydict$.

14. Write a function to invert a dictionary. It should accept a dictionary as a parameter and return a dictionary where the keys are values from the input dictionary and the values are lists of keys from the input dictionary. For example, this input:

```
{ "key1" : "value1", "key2" : "value2", "key3" : "value1" }
```

should return this dictionary:

```
{ "value1" : ["key1", "key3"], "value2" : ["key2"] }
```

15. Write a function called `word_frequencies`($mylist$) that accepts a list of strings called $mylist$ and returns a dictionary where the keys are the words from $mylist$ and the values are the number of times that word appears in $mylist$:

```
word_list = list("aaaaabbbbcccdde")
word_freq = { 'a' : 5, 'b' : 4, 'c' : 3, 'd' : 2, 'e' : 1 }

if word_frequencies(word_list) == word_freq :
    print("correct")
else :
    print("wrong")
```

16. In bioinformatics a k-mer is a substring of k characters from a string that is longer than k (see `https://en.wikipedia.org/wiki/K-mer` for details). Write a function with two parameters: a string containing DNA and the value of k. Return a dictionary of k-mer counts.