Please include your student number in filename (e.g. 1234567)

Student number:	2147105
Course title:	CS1PX
Questions answered:	ALL

1.a) func_1 takes 2 lists as an input. It iterates through the length of the lists and creates and returns a new dictionary, new_dict. The keys of the new dictionary are created from the elements in list_1 in normal order. The values that correspond to these keys are the elements in list_2 in reverse order.

b) list_1 = [12, 'g']
list_2 = [[2], 'b']

c) list_1 = ['a', 'b', 'c', 'c']
Adding a duplicate key into a dictionary will overwrite the original value with the new value, with only 1 instance of the key:value remaining present.

2.a) It is a higher order function. It iterates through the length of the given list. The variable 'i' takes on the index for each iteration. The has_correct_value function helps to check if the index equals the value at the point in the list, which will be True if the integers are consecutive. If False, the whole function returns the index 'i', which corresponds to the missing value. If always True, the function will finish by returning the final value of the list plus 1.

b) O(n). The function iterates through the whole list once, using a for loop. Adding 1 extra value to the list will add 1 extra iteration to the loop.

c) if lo > high:
    return lo

d) lo and high should be set at 0 and len(my_list)-1 to start. The function is recursive and higher order as it uses the has_correct_value function within it. It will then find the mid point of the list by floor division by 2 on high+lo. It then runs the has_correct_value function. If true, it will discard the first half of the list by setting the lo to mid+1, and calling itself again. If false, it will discard the last half of the list by setting the new high to mid-1. It repeats this until it reaches its base case, which will return lo, the index of the first missing element.

e) O(log n). Any given list is cut in half each time the recursive function calls itself. So a list double the size, would only take 1 extra call.

f)
```
def iterative_find_missing(my_list):
    lo, high = 0, len(my_list)-1
    while lo <= high:
        mid = (lo + high) // 2
        if has_correct_value(my_list, mid):
            lo = mid + 1
        else:
            high = mid - 1
    return lo
```

3.
```
def filter_dictionary(my_dict, keep_list):
    new_dict = {}
    try:
      if keep_list == []:
        print('keep_list is empty, please input a list of keys you wish to keep.')
        return {}
      for key in my_dict:
        if key in keep_list:
          new_dict[key] = my_dict[key]
      return new_dict
    except Exception as e:
      print('There has been the error: ' + str(e))
      return {}
```

4.a) my_list[i] = this_sum is a bug because it changes the original list.
It should be changed to new_list.append(this_sum)

this_sum = this_sum + j is a bug because it just sums the indexes of the list.
It should be changed to this_sum = this_sum + my_list[j]. The for loop is wrong too but that's another bug.

b) O(n^2). It has a nested for loop. Both loops iterate through the list. Adding 1 extra value can make the nested for loop iterate through the complete list again.

c)
```
def all_the_sums(my_list):
    new_list = []
    for i in range(len(my_list)):
      if i == 0:
        new_list.append(my_list[i])
      else:
        new_list.append(my_list[i] + new_list[i-1])
    return new_list
```

Complexity is O(n).

5.a) I will use a list of dictionaries, where each dictionary consists of 4 keys: name, speed, strength, size. The values of the keys are mapped to the corresponding values taken from the file. All integers apart from the name which is a string.

[{'name': 'POLLYBUB', 'speed': 8, 'strength': 3, 'size': 5 }, {'name': 'SUPASTRONG', 'speed': 2, 'strength': 10, 'size': 8 }]

b) Using my data structure:

```
def get_rating( monster_name, monster_group, attribute ):
        for dictionary in monster_group:
          if dictionary['name'] == monster_name:
            return dictionary[attribute]
        print('Error: no monster under that name found.')
        return None
```

c) Using my data structure:

```
def find_winner(monster_name_1, monster_name_2, monster_group, attribute):
        attribute_1 = get_rating( monster_name_1, monster_group, attribute)
        attribute_2 = get_rating( monster_name_2, monster_group, attribute)
        if attribute_1 > attribute_2:
          return monster_name_1
        elif attribute_1 < attribute_2:
            return monster_name_2
        else:
          return None
```

d)

```
def find_best(monster_group, attribute):
        rating = 0
        best = None
        for dictionary in monster_group:
          if dictionary['attribute'] > rating:
            rating = dictionary['attribute']
            best = dictionary['name']
        return best
```

e)