Lab 1

Custom Functions

```
def getDuration(start: time.time(), end: time.time()):
        return end - start

def getUniqueRandomNumbers(count: int, min:int, max:int):
        vals = []
        r = random.randint(min, max)
        for i in range(count):
        while r in vals:
        r = random.randint(min, max)
        vals.append(r)
        return vals

def getUniqueRandomDictKeys(count: int, d: dict):
        vals = []
        r = random.randint(0,len(d.keys()))
        for c in range(count):
        while r in vals or r not in d.keys():
        r = random.randint(0,len(d.keys()))
        vals.append(list(d.keys())[r])
        return vals

def printResults(title: str, list_times: [], dict_times: []):
        avg_list = sum(list_times)/len(list_times)
        avg_dict = sum(dict_times)/len(dict_times)
        print("\n"+title)
        print("\tList Times: "+str(list_times) + " | Average:
"+str(avg_list))
                print("\tDict Times: " + str(dict_times) + "
Average: "+str(avg_dict))

        if avg_list < avg_dict:
                print("\t\tFastest: List")
```

```
        else:
                print("\t\tFastest: Dictionary")
```

Generating List & Dictionary

```
"""
test_dict = {1:1,2:2,3:3,...}
test_list = {1,2,3,...}
"""

test_dict = dict()

for i in range(1000000):
        test_dict[i] = i

test_list = list(test_dict.keys())
```

Tests

Printing (Test 1):

```

"Print Testing"

list_print_times = []
dict_print_times = []

for i in range(3):
        start = time.time()
        print(test_list)
        end = time.time()
        list_print_times.append(getDuration(start,end))


for i in range(3):
        start = time.time()
        print(test_dict)
```

```
        end = time.time()
        dict_print_times.append(getDuration(start,end))
```

Output

```
Printing
        List Times: [0.5111091136932373, 0.2665238380432129,
0.26230597496032715] | Average: 0.3466463088989258
        Dict  Times: [0.5085949897766113, 0.5884931087493896,
0.5025579929351807] | Average: 0.5332153638203939
                Fastest: List
```

Table

	Trial 1	Trial 2	Trial 3
List	0.5111091136932373	0.2665238380432129	0.2665238380432129
Dict	0.5085949897766113	0.5884931087493896	0.5025579929351807

Retrieval (Test 2)

```
"""Retrieval Testing"""
list_retrieval_times = []
dict_retrieval_times = []

for i in range(3):
        r = random.randint(0,len(test_list))
        start=time.time()
        exists = r in test_list #performs a retrival and stores
existance of the object in the list as a boolean
        end=time.time()
        list_retrieval_times.append(getDuration(start,end))

for i in range(3):
        r = random.randint(0,len(test_dict))
        start=time.time()
```

```
        exists = r in test_dict #performs a retrival and stores existance of the object in the list as a boolean
        end=time.time()
dict_retrieval_times.append(getDuration(start,end))
```

Output

```
Retrieval
        List Times: [0.5111091136932373, 0.2665238380432129, 0.26230597496032715] | Average: 0.3466463088989258
        Dict  Times: [0.5085949897766113, 0.5884931087493896, 0.5025579929351807] | Average: 0.5332153638203939
                Fastest: List
```

Table

	Trial 1	Trial 2	Trial 3
List	0.5111091136932373	0.2665238380432129	0.2665238380432129
Dict	0.5085949897766113	0.5884931087493896	0.5025579929351807

Insertion (Test 3)

```
"""Insertion"""
list_insertion_times  = []
dict_inssertion_times = []


for i in range(3):
    r = random.randint(0,len(test_list))
    start=time.time()
    test_list.insert(r,r)
    end=time.time()
    list_insertion_times.append(getDuration(start,end))

for i in range(3):
```

```
    r = random.randint(0,len(test_dict))
    start=time.time()
    test_dict[r] = r
    end=time.time()
    dict_inssertion_times.append(getDuration(start,end))
```

Output

```
Insertion
        List Times: [0.0008230209350585938,
0.0007910728454589844, 0.0007619857788085938] | Average:
0.0007920265197753906
        Dict  Times: [2.1457672119140625e-06, 9.5367431640625e-
07, 9.2347411645622e-07] | Average: 1.2310386149088542e-06
               Fastest: Dictionary
```

Table

	Trial 1	Trial 2	Trial 3
List	0.0008230209350585938	0.0007910728454589844	0.0007619857788085
Dict	2.1457672119140625e-06	9.5367431640625e-07	9.2347411645622e-07

Deletion (Test 4)

```
#generating 3 unique random numbers to avoid trying to remove
items more than once
list_remove_indecies = getUniqueRandomNumbers(3, 1,
len(test_list))
dict_remove_keys = list_remove_indecies
print(list_remove_indecies)

list_remove_times = []
dict_remove_times = []
```

```
"Deletion"

for j in list_remove_indecies:
    start=time.time()
    del test_list[j]
    end=time.time()
    list_remove_times.append(getDuration(start,end))

for j in dict_remove_keys:
    if j in test_dict.keys():
        start = time.time()
        del test_dict[j]
        end=time.time()
        dict_remove_times.append(getDuration(start,end))
```

Output

```
Deletion
        List Times: [7.200241088867188e-05, 7.414817810058594e-
05, 0.0002219676971435547] | Average: 0.00012270609537760416
        Dict  Times: [2.1457672119140625e-06, 0.0,
9.5367431640625e-07] | Average: 1.0331471761067708e-06
               Fastest: Dictionary
```

Table

	Trial 1	Trial 2	Trial 3
List	7.200241088867188e-05	7.414817810058594e-05	0.0002219676971435547
Dict	2.1457672119140625e-06	0.0	9.2347411645622e-07

Print Functions

```
"output  section"
print("\n------------------------------------ Results --------
-------------------------------")
printResults("Printing", list_print_times, dict_print_times)
printResults("Retrieval", list_print_times, dict_print_times)
printResults("Insertion", list_insertion_times,
dict_inssertion_times)
printResults("Deletion", list_remove_times, dict_remove_times)
```

Questions

(1) What do you understand from this exercise? Explain in detail:
I learned that the list and dictionary data structures share different properties. Printing and retrieval take much less time for a list than a dictionary. This is due a list being linearly linked and involves searching through the entire list (in the worst case).
Insertion and deletion however favor dictionaries as they are not linked and only require a key to lookup the corresponding item. Adding to or deleting from a list involves modifying the index of each subsequent entry.