Lab 2

Lists:

Lists are what they seem - a list of values. Each one of them is numbered, starting from zero. You can remove values from the list, and add new values to the end. Example: Your many cats' names. *Compound* data types, used to group together other values. The most versatile is the *list*, which can be written as a list of comma-separated values (items) between square brackets. List items need not all have the same type.

```
cats = ['Tom', 'Snappy', 'Kitty', 'Jessie', 'Chester']

print (cats[2]) cats.append('Oscar')

#Remove 2nd cat, Snappy.del cats[1]
```

Compound datatype:

```
>>> a = ['spam', 'eggs', 100, 1234]

>>> a[1:-1]    #start at element at index 1, end before last element
['eggs', 100]
>>> a[:2] + ['bacon', 2*2]
['spam', 'eggs', 'bacon', 4]
>>> 3*a[:3] + ['Boo!']
['spam', 'eggs', 100, 'spam', 'eggs', 100, 'spam', 'eggs', 100, 'Boo!']

>>> a= ['spam', 'eggs', 100, 1234]
>>> a[2] = a[2] + 23
>>> a
['spam', 'eggs', 123, 1234]
```

Replace some items:

```
>>> a[0:2] = [1, 12]
>>> a
[1, 12, 123, 1234]
```

Remove some:

```
>>> a[0:2] = []
>>> a
[123, 1234]
```

Clear the list: replace all items with an empty list:

```
>>> a[:] = []
>>> a
[]
```

Length of list:

```
>>> a = ['a', 'b', 'c', 'd']
>>> len(a)
4
```

Nested lists:

```
>>> q = [2, 3]
>>> p = [1, q, 4]
>>> len(p)
3
>>> p[1]
[2, 3]
```

Functions of lists:

list.append(x): Add an item to the end of the list; equivalent to a[len(a):] = [x]**list.extend(L):** Extend the list by appending all the items in the given list; equivalent toa[len(a):] = L.

list.insert(i, x): Insert an item at a given position. The first argument is the index of theelement before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).

list.remove(x): Remove the first item from the list whose value is x. It is an error if there isno such item.

list.pop(i): Remove the item at the given position in the list, and return it. If no index isspecified, a.pop() removes and returns the last item in the list.

list.count(x): Return the number of times x appears in the list.

list.sort(): Sort the items of the list, in place.

list.reverse(): Reverse the elements of the list, in place.

Tuples:

Tuples are just like lists, but you can't change their values. Again, each value is numberedstarting from zero, for easy reference. Example: the names of the months of the year.

months = ('January' , 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September','October', 'November', 'December')

Index	Value
0	January
1	February
2	March
3	April
4	May
5	June
6	July
7	August
8	September
9	October
10	November
11	December

We can have easy membership tests in Tuples using the keyword in.

```
>>> 'December' in months   # fast membership testingTrue
```

Sets:

A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Curly braces or the set() function can be used to create sets. Note: to create an empty set you have to use set(), not {}; the latter creates an empty dictionary.

```
Example 1:

>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']

>>> fruit = set(basket)  # create a set without duplicates

>>> fruit

{'banana', 'orange', 'pear', 'apple' }

>>> 'orange' in fruit   # fast membership testingTrue

>>> 'crabgrass' in fruitFalse

Example 2:

>>> # Demonstrate set operations on unique letters from two words

>>> a = set('abracadabra')

>>> b = set('alacazam')

>>> a                                  # unique letters in a

{'a', 'r', 'b', 'c', 'd'}

>>> a - b                              # letters in a but not in b

{'r', 'd', 'b'}

>>> a | b                              # letters in either a or b

{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}

>>> a & b                              # letters in both a and b

{'a', 'c'}

>>> a ^ b                              # letters in a or b but not both

{'r', 'd', 'b', 'm', 'z', 'l'}
```

Set comprehensions are also supported:

```
>>> a = {x for x in 'abracadabra' if x not in 'abc'}

>>> a

{'r', 'd'}
```

Dictionaries:

Dictionaries are similar to what their name suggests - a dictionary. In a dictionary, you have an 'index' of words, and for each of them a definition.

In python, the word is called a 'key', and the definition a 'value'. The values in a dictionary aren't numbered - they aren't in any specific order, either - the key does the same thing.

You can add, remove, and modify the values in dictionaries. Example: telephone book.

The main operations on a dictionary are storing a value with some key and extracting the value given the key. It is also possible to delete a key:value pair with `del`. If you store using a key that is already in use, the old value associated with that key is forgotten. It is an error toextract a value using a non-existent key.

Performing list(d.keys()) on a dictionary returns a list of all the keys used in the dictionary, inarbitrary order (if you want it sorted, just use sorted(d.keys()) instead). To check whether a single key is in the dictionary, use the **in** keyword.

At one time, only one value may be stored against a particular key. Storing a new value for a existing key overwrites its old value. If you need to store more than one value for a particular key, it can be done by storing a list as the value for a key.

```
phonebook = {'ali':8806336, 'omer':6784346,'shoaib':7658344,'saad':1122345}

#Add the person '' to the phonebook:phonebook['waqas'] = 1234567 print("Original
Phonebook") print(phonebook)

# Remove the person 'shoaib' from the phonebook:del phonebook['shoaib']


print("'shoaib' deleted from phonebook")print(phonebook)


phonebook = {'Andrew Parson':8806336, \

'Emily Everett':6784346, 'Peter Power':7658344, \'Louis Lane':1122345} print("New
phonebook")print(phonebook)


#Add the person 'Gingerbread Man' to the phonebook:phonebook['Gingerbread Man'] =
1234567


list(phonebook.keys()) sorted(phonebook.keys()) print( 'waqas' in phonebook)

print( 'Emily Everett' in phonebook)

#Delete the person 'Gingerbread Man' from the phonebook:del phonebook['Gingerbread
Man']
```

Exercises

1. Create a movies list containing a single tuple. The tuple should contain a movie title, the director's name, the release year of the movie, and the movie's budget.

   Sol: movies = [("The Room", "Tommy Wiseau", "2003", "$6,000,000")]

2. Use the input function to gather information about another movie. You need a title, director's name, release year, and budget.

   Sol: movies = [("The Room", "Tommy Wiseau", "2003", "$6,000,000")]

   title = input("Title: ")

   director = input("Director: ")

   year = input("Year of release: ")

   budget = input("Budget: ")

3. Create a new tuple from the values you gathered using input. Make sure they're in the same order as the tuple you wrote in the movies list.

   Sol: movies = [("The Room", "Tommy Wiseau", "2003", "$6,000,000")]

   title = input("Title: ")

   director = input("Director: ")

   year = input("Year of release: ")

   budget = input("Budget: ")

   new_movie = title, director, year, budget

4. Use an f-string to print the movie name and release year by accessing your new movie tuple.

   Sol: title = input("Title: ")

   director = input("Director: ")

   year = input("Year of release: ")

   budget = input("Budget: ")

   new_movie = title, director, year, budget

   print(f"{new_movie[0]} ({new_movie[2]})")

5. Add the new movie tuple to the movies collection using append.

   Sol: Since we already created our new_movie variable, which refers to a tuple, this step is fairly straightforward. We just need to call append using the dot syntax, and we need to pass in new_movie when we call the method:

   movies = [("The Room", "Tommy Wiseau", "2003", "$6,000,000")]

   title = input("Title: ")

   director = input("Director: ")

   year = input("Year of release: ")

   budget = input("Budget: ")

   new_movie = title, director, year, budget

```
print(f"{new_movie[0]} ({new_movie[2]})")

movies.append(new_movie)
```

6. Print both movies in the movies collection.

Sol:
```
movies = [("The Room", "Tommy Wiseau", "2003", "$6,000,000")]
title = input("Title: ")
director = input("Director: ")
year = input("Year of release: ")
budget = input("Budget: ")
new_movie = title, director, year, budget
print(f"{new_movie[0]} ({new_movie[2]})")
movies.append(new_movie)
print(movies[0])
print(movies[1])
```

7. Remove the first movie from movies. Use any method you like.

Sol: We have a lot of options here.
First we could use del like so:
```
del movies[0]
```
Alternatively, we could use pop:
```
movies.pop(0)
```
pop is a method, so we need to use this dot syntax.
Alternatively we could use the remove method, passing in the item at index 0:
```
movies.remove(movies[0])
```
I think del is the cleanest option in this case, so my finished solution looks like this:
```
movies = [("The Room", "Tommy Wiseau", "2003", "$6,000,000")]
title = input("Title: ")
director = input("Director: ")
year = input("Year of release: ")
budget = input("Budget: ")
new_movie = title, director, year, budget
print(f"{new_movie[0]} ({new_movie[2]})")
movies.append(new_movie)
print(movies[0])
print(movies[1])
del movies[0]
```

Lab 2 Task:

MiniProject 1:

Today's project is actually a very common interview question, which revolves around a childhood counting game called Fizz Buzz.

In case you're not familiar with the game, it goes like this:

One player starts by saying the number 1.
Each player then takes it in turns to say the next number, counting one at a time.
If the number is divisible by 3, instead of saying the number, the player should say, "Fizz".
If the number is divisible by 5, instead of saying the number, the player should say, "Buzz".
If the number is divisible by 3and5, instead of saying the number, the player should say, "Fizz Buzz".
If you make a mistake, you're usually eliminated from the game, and the game continues until there's only a single player remaining.
If there are no mistakes, the first 15 rounds of Fizz Buzz should look like this:

1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
Fizz Buzz

Below you'll find a brief explaining what to do for our version, and you'll also find a model solution with an accompanying explanation. I'd really recommend you try to do this on your own before checking out our version.

Just like with the day 3 project, there's nothing wrong with looking back at the content for the last 6 days, or referencing your notes. You also shouldn't be worried if your solution is a little different to ours, as there are many, many ways to tackle this particular problem.

The brief
For our version, we're only going to have a single player, the computer, and it's going to play the first 100 rounds of Fizz Buzz all by itself. In other words, we need to print out the first 100 items in the sequence, starting from 1.

In order to complete this exercise, you're going to need to use loops, and you can generate your list of numbers using range. You're also going to need conditionals, and you're going to need be able to check if something is divisible by 3 or 5.

For this last part, you can use an operator called modulo, which uses the percent symbol (%). Modulo will give you the remainder of a division, so if a number is divisible by 3, the value of number % 3 will be 0.

MiniProject 2

In particular we're going to finding the average budget of the films in our data set, and we're going to identify high budget films that exceed the average budget we calculate.

The brief

Below you'll find a list which contains the relevant data about a selection of movies. Each item in the list is a tuple containing a movie name and movie budget in that order:

```
movies = [
    ("Eternal Sunshine of the Spotless Mind", 20000000),
    ("Memento", 9000000),
    ("Requiem for a Dream", 4500000),
    ("Pirates of the Caribbean: On Stranger Tides", 379000000),
    ("Avengers: Age of Ultron", 365000000),
    ("Avengers: Endgame", 356000000),
    ("Incredibles 2", 200000000)
]
```

For this project, your program should do the following:

Calculate the average budget of all movies in the data set.

Print out every movie that has a budget higher than the average you calculated. You should also print out how much higher than the average the movie's budget was.

Print out how many movies spent more than the average you calculated.

If you want a little extra challenge, allow users to add more movies to the data set before running the calculations.

You can do this by asking the user how many movies they want to add, which will allow you to use a for loop and range to repeat some code a given number of times. Inside the for loop, you can write some code that takes in some user input and appends a movie tuple containing the collected data to the movie list.