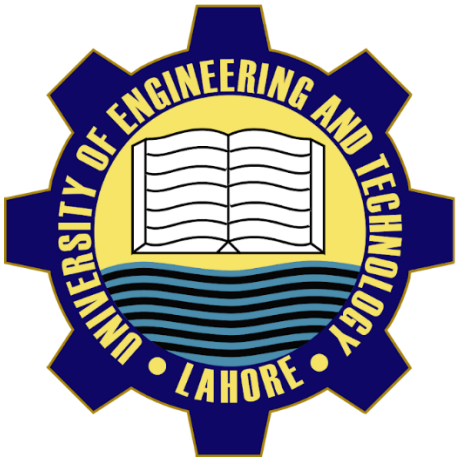
MCT-242 : COMPUTER PROGRAMMING-I

using Python 3.9



Prepared By:

Mr. Muhammad Ahsan Naeem

YouTube Playlist

https://www.youtube.com/playlist?list=PLWF9TXck7O_wMDB-VriREZ6EvwkWLNB7q

LAB 24: DICTIONARY: CLO-4

Dictionary in Python is a more general form of a list where to access an element, instead of using the index, we can use a key. Every element in a dictionary is a **key-value** pair. The **key** can be of any data type and not just the numbers as indices. The difference between simple list and dictionary is explained here:

We can define a list to hold number of days in each month as:

```
days=[31,28,31,30,31,30,31,31,30,31,30,31]
```

Later, if we want to access days of January, we must know that it is at index 0. Hence we will access that as `dasy[0]` and so on for any other month.

If we want to do the same thing using dictionary, we can define it as 12 key-value pairs:

```
days = {'Jan':31,'Feb':28,'Mar':31,'Apr':30,'May':31,'Jun':30,'Jul':3
1,'Aug':31,'Sep':30,'Oct':31,'Nov':30,'Dec':31}
```

To make it more readable we can have:

```
days = {
    'Jan':31,
    'Feb':28,
    'Mar':31,
    'Apr':30,
    'May':31,
    'Jun':30,
    'Jul':31,
    'Aug':31,
    'Sep':30,
    'Oct':31,
    'Nov':30,
    'Dec':31
    }
```

Here `days` is a dictionary having 12 elements (key-value pair). So, if we want to access the days of April for example, we don't need to know the position within the list, rather we can access that through its key as here:

```
print(days['Apr'])
```

A simple comparison between a list and dictionary is given here:

List	Dictionary
Objects retrieved by location (index)	Objects retrieved by key name
Ordered sequence	Unordered and cannot be sorted
Mutable	Mutable

Key of a Dictionary:

Keys are unique within a dictionary while **values** need not to be, as in above example, multiple entries have value 30 or 31 but no two keys are same. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples, but generally keys are strings. You can mix different types of keys in the same dictionary and different types of values, too.

Example:

Let's create a dictionary having different attributes of one student as:

```
student1={'fName':'Ahmad','lName':'Hassan','courses':['CP','ED','EM']
}
print(student1['courses'])
```

You can see that one key of above dictionary `'courses'` has a list as its value.

If we try to access a key which is not defined in dictionary it generates an error as the below statement will do for above dictionary:

```
print(student1['email'])
```

KeyError: 'email'

Errors are the nightmares for programmers. So, it always better to use `get()` method of dictionary class to access a value from key. If the key doesn't exist, this method will return `None` instead of generating error as shown here:

```
print(student1.get('email'))
```

We can also return some customized message instead of `None` by passing that message as second argument of get method as shown here:

```
print(student1.get('email','Key not found'))
```

If key is a valid key `get()` method will return the value instead of the customized message.

Once a dictionary is defined, we can add more key-values (and hence dictionary is mutable) as shown here:

```
student1={'fName':'Ahmad','lName':'Hassan','courses':['CP','ED','EM']
}
student1['email']='ahmad.hassan@gmail.com'
print(student1.get('email','Key not found'))
```

Now the above code will display `ahmad.hassan@gmail.com`

The above method can also be used to update the value of a particular key as shown here:

```
student1={'fName':'Ahmad','lName':'Hassan','courses':['CP','ED','EM']
}
student1['courses']=['ED','EM']
```

```
print(student1.get('courses','Key not found'))
```

The above code will update the value of courses in second line.

There is a method in Dictionary class called **update()** that can be used to add or update one or multiple keys/values in one statement. The use is shown here:

```
student1={'fName':'Ahmad','lName':'Hassan','courses':['CP','ED','EM']
}
student1.update({'courses':['ED','EM'],'fName':'Muhammad Ahmad','emai
l':'ahmad.hassan@gmail.com'})
print(student1)
```

See carefully that in **update()** method if a new key is passed, that key/value will be added in the dictionary and if old key is passed the previous value of that key will be updated.

Dictionary Class Methods:

Method	Description
copy()	Returns a shallow copy of the dictionary.
clear()	Removes all items from the dictionary.
pop(k)	Removes and returns an element from a dictionary having the given key **k**.
popitem()	Removes the arbitrary key-value pair from the dictionary and returns it as tuple. Before Python 3.7, arbitrary meant random pair but from Python 3.7 it refers to last entered key.
get(k)	It is a conventional method to access a value for a key **k**.
values()	Returns a list of all the values available in a given dictionary
setdefault(k,v)	Returns the value of a key **k** (if the key is in dictionary). If not, it inserts key **k** with a value **v** to the dictionary
keys()	Returns list of all dictionary keys
items()	Returns a list of dictionary (key, value) tuple pairs

Process whole dictionary:

We can process a complete dictionary using a **for** loop and **in** operator as we do in case of lists and strings. But this method will loop through the **keys** only. See the code below:

```
student1={'fName':'Ahmad','lName':'Hassan','courses':['CP','ED','EM']
}
for i in student1:
    print(i)
```

Here is the output of the above code:

```
fName
lName
```

```
courses
```

If we want to access the **values** instead of the **keys** we should use `values()` method in `for` loop as shown here:

```
student1={'fName':'Ahmad','lName':'Hassan','courses':['CP','ED','EM']
}
for i in student1.values():
    print(i)
```

We can also access both keys and values in one loop using `items()` method as shown here:

```
student1={'fName':'Ahmad','lName':'Hassan','courses':['CP','ED','EM']
}
for i,j in student1.items():
    print(i,j)
```

Using `items()` method, both the keys and values are accessed in `for` loop. These are accessed as two-valued tuples in each iteration. In above code, the tuples are unpacked. But we could have this code:

```
student1={'fName':'Ahmad','lName':'Hassan','courses':['CP','ED','EM']
}
for i in student1.items():
    print(i)
```

It will have the output as:

```
('fName', 'Ahmad')
('lName', 'Hassan')
('courses', ['CP', 'ED', 'EM'])
```

Tasks:

[1] A dry-fruits seller has variety of products with different price rates. We can define a dictionary with **key** as name of dry-fruit and **value** as its price per KG. Here is this dictionary with few pairs:

```
products={'walnuts':1500,'cashew':1800,'almond':2000,'pine nuts'
:8000}
```

Write a program that will do the followings for the seller support:

- If user enters a product name, it should display its rate. If the entered product doesn't exist, it should display proper message instead of error.
- If user wants to add new product or update price of existing products, he should be able to do that.
- If user wants to serve a customer, the program must ask for the product and amount (in KG) and at the end, it must generate the bill stating each item purchased (product and quantity) and the total bill.

The program must ask to select one out of above three modes.

[2] Do the above task by creating three functions for three modes.

[3] Use the dictionary of above task with a few entries (at least 7). The program should ask for an amount to enter and will display all products (not rates) with price less than or equal to that.

Empty Dictionary:

Like empty list we can have empty dictionary defined as:

```
a={}
```

The use of empty dictionary is explained in following example:

Example:

Write a program that will take a sentence from user and will display count of each alphabet **(a-z)** in that sentence.

We can do it without dictionary and can store the count in a list of size 26, but we will have to write 26 `if` statements for that. On the other hand, using dictionary, we can read one character and add it to dictionary as **key** (if it's first time occurrence of that character) and increment the **value** of that key by one (if the character already exists as key). Here is the code:

```
msg=input("Enter a Sentence:") #input message
alphacount = {} #starting with empty dictionary
for n in msg.lower(): #looping over message characters
    if n not in alphacount.keys(): #character appeared first time
        alphacount[n] = 1
    else: #if that character already exists
        alphacount[n] += 1
print(alphacount)
```

A sample output is shown here:

```
Enter a Sentence: A simple sentence to test the code
{'a': 1, ' ': 6, 's': 3, 'i': 1, 'm': 1, 'p': 1, 'l': 1, 'e': 7, 'n':
2, 't': 5, 'c': 2, 'o': 2, 'h': 1, 'd': 1}
```

We can upgrade above code for a couple of features as explained below:

Firstly, the space is also displayed which is not really required. We can remove all spaces from the message as:

```
msg=msg.replace(" ","")
```

The same can be done for the period (.) and in general if we want to exclude all punctuation symbols, we can use a string constant named as `punctuation` in `string` module (like `pi` in `math` module) that contains all punctuation symbols but not the space. To exclude all punctuation marks we can add this part:

```
import string
for p in string.punctuation:
    msg=msg.replace(p,"")
```

Secondly, it will be appropriate to display results alphabetically instead of the first occurrence order. Dictionaries cannot be sorted but the list can be. We can use `list()` method to convert dictionary into a list and then sort that list. The `list()` method applied on a dictionary creates a list of tuples with key-value pairs. Complete code incorporating above two features is given as:

```
import string
msg=input("Enter a Sentence: ") #input message
msg=msg.replace(" ","")
for p in string.punctuation:
    msg=msg.replace(p,"")
alphacount = {} #starting with empty dictionary
for n in msg.lower(): #looping over message characters
    if n not in alphacount.keys(): #character appeared first time
        alphacount[n] = 1
    else: #if that character already exists
        alphacount[n] += 1

pairs = list(alphacount.items())
pairs.sort()
for i in pairs:
    print(i)
```

Finally, what if we want to display result not alphabetically but in order of frequency of each alphabet? In such case we can reverse the order of the tuple so that it has first value is count and the second is alphabet as:

```
pairs = list(alphacount.items())
items = [(x, y) for (y, x) in pairs]
items.sort(reverse=True)
```

Another quite useful point is if you are not willing to work on Tuple, you can convert them into list with two elements. To do that, all you need is to replace following line:

```
items = [(x, y) for (y, x) in pairs]
```

with this one:

```
items = [[x, y] for (y, x) in pairs]
```

Tasks:

[4] Very similar to the above example, write a program that will find the frequencies of occurrence of each word in a string and will display them in order of their frequencies.

[5] The login details of users on some platform are stored as Dictionary here:

```
logins={'ahsan':'abc123','naeem':'asd345','ali':'hello'}
```

You should have at least 10-15 entries in above Dictionary. Now write a code that should ask the user to enter their username and password. If the username is not in the dictionary, the program should indicate that the person is not a valid user of the system. If the username is in the dictionary, but the user did not enter the right password, the program should say that the password is invalid. If the password is correct, then the program should tell the user that they are now logged in to the system.

2-D Dictionary or List of Dictionary?

If we have to create a dictionary of all students of a class with **key** as registration number and **value** as student's name, we can do it as:

```
session={
    '2018-MC-01':'Muhammad Usman',
    '2018-MC-02':'Tahir Mehmood',
    '2018-MC-03':'Muhammad Bilal'
    }
```

We can add as many students as we want in above dictionary, but it just contains the information of registration number and name. What if we need to store more information about one student? Like the section and the list of subjects registered? This information for one student can be stored as dictionary as shown here:

```
student1={
    'Reg':'2018-MC-01',
    'Name':'Muhammad Usman',
    'Sec':'A',
    'Courses':['CP','ED','EM']}
```

But what if we want to store such information for the whole class? For this we can have a 2-D dictionary with registration number as **key** and a dictionary (having other info) as **value**. Second option is to have a list with each element as dictionary of student. It's all about personal preference, but I think using a list of dictionary is better. Here is an example list having three students as dictionary:

```
student1={
    'Reg':'2018-MC-01',
    'Name':'Muhammad Usman',
    'Sec':'A',
    'Courses':['CP','ED','EM']}
student2={
    'Reg':'2018-MC-02',
    'Name':'Tahir Mehmood',
    'Sec':'A',
    'Courses':['CP','DLD','EM']}
student3={
    'Reg':'2018-MC-03',
```

```
    'Name':'Muhammad Bilal',
    'Sec':'A',
    'Courses':['VCA','ED','EM']}
allStudents=[student1,student2,student3]
```

Later, if we have to access any student, we can do it using list index as:

```
print(allStudents[2])
```

However, if you want to create a 2D dictionary, the **key** can be the registration number and the **value** can be another dictionary with other detail of the student.

```
allStudents={
    '2018-MC-01': {'Name':'Muhammad Usman','Sec':'A','Courses':['CP','ED','EM']},
    '2018-MC-02': {'Name':'Tahir Mehmood','Sec':'A','Courses':['CP','DLD','EM']},
    '2018-MC-03': {'Name':'Muhammad Bilal','Sec':'A','Courses':['VCA','ED','EM']}
}
```

So, to access one student detail, we can access that using the key as the registration number:

```
print(allStudents['2018-MC-02'])
```

And suppose if you want to access the courses list of that student, you can do it like:

```
print(allStudents['2018-MC-02']['Courses'])
```

Dictionary Comprehension:

Like list comprehension and the set comprehension, we have the Dictionary Comprehension. Suppose we want to create a Dictionary such that the keys are the integers from 1 to 10 and the values are the squares of those numbers. We can do that as:

```
d={i:i*i for i in range(11)}
print(d)
```

Many times we have a case to create a dictionary from two other iterables, in a way that the keys should be the elements of one iterable and the values should be the elements of the second iterable on corresponding indices. We know we can use the **`zip`** function to zip multiple iterbales together and then we can use the Dictionary Comprehension on that. But in such case, even a simple solution is using the **`dict()`** constructor on the zip object and that will return the desired dictionary.

```
gLetters=['A+','A','A-','B+','B','B-','C+','C','C-','D+','D','F']
gPoints=[4,4,3.7,3.3,3.0,2.7,2.3,2.0,1.7,1.3,1.0,0.0]
gDict=dict(zip(gLetters,gPoints))
```

```
gDict={k:v for k,v in zip(gLetters,gPoints)}
print(gDict)
```

Tasks:

[6] Create a User-Defined function named as **`calculateGPA`** that will take two lists; one as a list of **Credit Hours** of different subjects and the second as the list of **Letter Grades** obtained in those courses by some student. Use the Dictionary for grade mapping as give above and the function must return the **GPA** of that student which is calculated as:

$$GPA = \frac{\sum(GP_i * CH_i)}{\sum CH_i}$$

[7] **Morse code** is an encoding scheme that uses dashes and dots to represent numbers and letters. In this exercise, you will write a program that uses a dictionary to store the mapping from letters and numbers to Morse code. Use a period to represent a dot, and a hyphen to represent a dash. The mapping from letters and numbers to dashes and dots is shown in Table. Your program should read a message from the user. Then it should translate each letter and number in the message to Morse code by creating a function names as **`MorseCodeGenerator`**. Your program should keep the space unchanged and you can assume there is not any other special character in the input string.

Letter	Code	Letter	Code	Letter	Code	Number	Code
A	.-	J	.- - -	S	...	1	.- - - -
B	-...	K	-.-	T	-	2	..- - -
C	-.-.	L	.-..	U	..-	3	...- -
D	-..	M	- -	V	...-	4	....-
E	.	N	-.	W	.- -	5	.....
F	..-.	O	- - -	X	-..-	6	-....
G	- -.	P	.- -.	Y	-.- -	7	- -...
H	....	Q	- -.-	Z	- -..	8	- - -..
I	..	R	.-.	0	- - - - -	9	- - - -.