



SARASWATI Education Society's  
**SARASWATI College of Engineering**  
Learn Live Achieve and Contribute  
Kharghar, Navi Mumbai - 410 210.  
**NAAC Accredited**

## **Department of Computer Engineering**

**Name: Shaikh Adnan Shaukat Ali**

**Rno: 55**

**Subject: Python Programming**

**Class/Sem: SE/IV**



**Department of Computer Engineering**

# **INDEX**

**Subject :Python Programming**

**Class/Sem: SE/IV**

<b>Expt. No.</b>	<b>Name of the Experiment</b>	<b>Page No</b>
1	Exploring basics of python like data types (strings, list, array, dictionaries, set, tuples) and control statements.	1-18
2	Creating functions, classes and objects using python. Demonstrate exception handling and inheritance.	19-27
3	Exploring Files and directories a. Python program to append data to existing file and then display the entire file b. Python program to count number of lines, words and characters in a file. c. Python program to display file available in current directory	28-31
4	Creating GUI with python containing widgets such as labels, textbox, radio, checkboxes and custom dialog boxes.	32-40
5	Menu driven program for data structure using built in function for link list, stack and queue.	41-44
6	Program to demonstrate CRUD ( <b>create, read, update and delete</b> ) operations on database (SQLite/ MySQL) using python.	45-47
7	Creation of simple socket for basic information exchange between server and client.	48-51
8	Programs on Threading using python.	52-53
9	Exploring basics of NumPy Methods.	53-56
10	Program to demonstrate use of NumPy: Array objects.	57-58
11	Program to demonstrate Data Series and Data Frames using Pandas.	59-60
12	Program to send email and read content of URL.	61-63

**Sonali Shukla**  
**Subject I/C**



SARASWATI Education Society's  
**SARASWATI College of Engineering**

Learn Live Achieve and Contribute

Kharghar, Navi Mumbai - 410 210.

**NAAC Accredited**

## Department of Computer Engineering

**Subject : Python Programming**

**Class/Sem:SE/IV**

### INDEX

Expt. No.	Name of the Experiment	Page No
1	Assignment 1	64-66
2	Assignment 2	67-70

**Sonali Shukla**  
**Subject I/C**

## EXPERIMENT NO. 1

**Aim :** Exploring basics of python like data types (strings, list, array, dictionaries, set, tuples) and control statements.

**Resources Required:** Consumables – Printer Pages for printouts.

### Theory:

Lists:

Lists are positionally ordered collections of arbitrarily typed objects, and they have no fixed size and they are mutable. Lists are contained in square brackets []. Lists can contain numbers, strings, nested sublists, or nothing

Examples: L1 = [0,1,2,3],

L2 = ['zero', 'one'],

L3 = [0,1,[2,3],'three',['four,one']], L4 = [].

List indexing works just like string indexing. Lists are mutable: individual elements can be reassigned in place. Moreover, they can grow and shrink in place.

Example:

```
>>> L1 = [0,1,2,3]
```

```
>>> L1[0] = 4
```

```
>>> L1[0] 4
```

**Basic List Operations** Lists respond to the + and \* operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string. Some of basic operations of list are as follows.

Python includes following list methods:

Sr.No	Methods	Description
1	list.append(obj)	Appends object obj to list
2	list.count(obj)	Returns count of how many times obj occurs in list
3	list.extend(seq)	Appends the contents of seq to list
4	list.index(obj)	Returns the lowest index in list that obj appears
5	list.insert(index, obj)	Inserts object obj into list at offset index
6	list.pop(obj=list[-1])	Removes and returns last object or obj from list

7	<code>list.remove(obj)</code>	Removes object obj from list
8	<code>list.reverse()</code>	Reverses objects of list in place

Tuples:

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets. Creating a tuple is as simple as putting different comma-separated values.

Optionally you can put these comma-separated values between parentheses also.

For example

```
tup1 = ('physics', 'chemistry', 1997, 2000);
```

```
tup2 = (1, 2, 3, 4, 5 );
```

```
tup3 = "a", "b", "c", "d";
```

Advantages of Tuple over List

However, there are certain advantages of implementing a tuple over a list. Below listed are some of the main advantages:

- We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.
- Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
- Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

## 1. Creating a Tuple

A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma. The parentheses are optional but is a good practice to write it. A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).

## 2. Accessing Elements in a Tuple

There are various ways in which we can access the elements of a tuple.

### a. Indexing

We can use the index operator [] to access an item in a tuple where the index starts from 0. So, a tuple having 6 elements will have index from 0 to 5. Trying to access an element other than (6, 7,...) will raise an IndexError. The index must be an integer, so we cannot use float or other types. This will result into TypeError.

### b. Negative Indexing

Python allows negative indexing for its sequences. The index of

-1 refers to the last item, -2 to the second last item and so on.

### c. Changing a Tuple

Unlike lists, tuples are immutable. This means that elements of a tuple cannot be changed once it has been assigned. But, if the element is itself a mutable datatype like list, its nested items can be changed.

### d. Python Tuple Methods

Methods that add items or remove items are not available with tuple. Only the following two methods are available

Method	Description
count(x)	Return the number of items that is equal to x
index(x)	Return index of first item that is equal to x

### Sets:

A Set is an unordered collection data type that is iterable, mutable and has no duplicate elements. Python's set class represents the mathematical notion of a set. The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set. This is based on a data structure known as a [hash table](#).

### Methods for Sets:

1. **add(x) Method:** Adds the item x to set if it is not already present in the set.
2. **union(s) Method:** Returns a union of two set. Using the '|' operator between 2 sets is the same as writing set1.union(set2)
3. **intersect(s) Method:** Returns an intersection of two sets. The '&' operator comes can also be used in this case.
4. **difference(s) Method:** Returns a set containing all the elements of invoking set but not of the second set. We can use '-' operator here.

**clear() Method:** Empties the whole set.

### Dictionary:

Dictionary in python consists of keys and their values.

1. **Create a Python Dictionary** Here is an example of how we can create a dictionary in Python :

```
>>> myDict = {"A":"Apple", "B":"Boy", "C":"Cat"}
```

In the above example: A dictionary is created. This dictionary contains three elements. Each element constitutes of a key value pair. This dictionary can be accessed using the variable myDict.

2. **Access Dictionary Elements** Once a dictionary is created, you can access it using the variable to which it is assigned during creation. For example, in our case, the variable myDict can be used to access the dictionary elements. Here is how this can be done :

```
>>> myDict["A"] 'Apple'
```

```
>>> myDict["B"] 'Boy'
```

```
>>> myDict["C"] 'Cat'
```

So you can see that using the variable myDict and Key as index, the value of corresponding key can be accessed. For those who have C/C++ background, it's more like accessing the value kept at a particular index in an array. If you just type the name of the variable myDict, all the key value pairs in the dictionary will be printed.

```
>>> myDict {'A': 'Apple', 'C': 'Cat', 'B': 'Boy'}
```

Only dictionary keys can be used as indexes. This means that myDict["A"] would produce 'Apple' in output but myDict["Apple"] cannot produce 'A' in the output.

```
>>> myDict["Apple"]
```

```
Traceback (most recent call last): File "<stdin>", line 1, in
```

```
<module> KeyError: 'Apple' So we see that the python compiler complained about 'Apple' being used as index.
```

3. Update Dictionary Elements Just the way dictionary values are accessed using keys, the values can also be modified using the dictionary keys. Here is an example to modify python dictionary element:

```
>>> myDict["A"] = "Application"
```

```
>>> myDict["A"] 'Application'
```

```
>>> myDict
```

```
{'A': 'Application', 'C': 'Cat', 'B': 'Boy'}
```

You can see that in the example shown above, the value of key 'A' was changed from 'Apple' to 'Application' easily. This way we can easily conclude that there could not be two keys with same name in a dictionary.

4. Delete Dictionary Elements Individual elements can be deleted easily from a dictionary. Here is an example to remove an element from dictionary.

```
>>> myDict {'A': 'Application', 'C': 'Cat', 'B': 'Boy'}
```

```
>>> del myDict["A"]
```

```
>>> myDict
```

```
{'C': 'Cat', 'B': 'Boy'}
```

So you can see that by using 'del' an element can easily be deleted from the dictionary.

If you want to delete complete dictionary i.e all the elements in the dictionary then it can be done using the clear() function. Here is an example :

```
>>> myDict {'C': 'Cat', 'B': 'Boy'}
```

```
>>> myDict.clear()
```

```
>>> myDict
```

```
{}
```

So you see that all the elements were deleted making the dictionary empty.

#### 1. Dictionaries are Unordered

```
>>> myDict = {"A": "Apple", "B": "Boy", "C": "Cat"}
```

```
>>> myDict
```

```
{'A': 'Apple', 'C': 'Cat', 'B': 'Boy'}
```

You can observe that the order of elements while the dictionary was being created is different from the order

in which they are actually stored and displayed.

Even if you try to add other elements to python dictionary:

```
>>> myDict["D"] = "Dog"
```

```
>>> myDict
```

```
{'A': 'Apple', 'C': 'Cat', 'B': 'Boy', 'D': 'Dog'}
```

```
>>> myDict["E"] = "Elephant"
```

```
>>> myDict
```

```
{'A': 'Apple', 'C': 'Cat', 'B': 'Boy', 'E': 'Elephant', 'D': 'Dog'}
```

You'll observe that it's not necessary that elements will be stored in the same order in which they were created.

2. Dictionary Keys are Case Sensitive. Same key name but with different case are treated as different keys in python dictionaries. Here is an example :

```
>>> myDict["F"] = "Fan"
```

```
>>> myDict["f"] = "freeze"
```

```
>>> myDict
```

```
{'A': 'Apple', 'C': 'Cat', 'B': 'Boy', 'E': 'Elephant', 'D': 'Dog',  
'F': 'Fan', 'f': 'freeze'}
```

**Conclusion:** Hence, we have successfully study concept of Strings, Set, List, Tuples, Dictionary and control statement in python.

**List:**

**Code:**

```
def even_odd(l = []):
```

```
    even, odd = [], []
```

```
    for x in l:
```

```
        if x%2 == 0:
```

```
            even.append(x)
```

```
        else:
```

```
            odd.append(x)
```

```
    return even,odd
```

```
def merge_sort(l1 = [],l2 = []):
```

```
    lis = [*l1,*l2]
```

```
    return sorted(lis)
```



```

def update_delete(l = []):
    l[0] = input("Enter First value to update: ")
    del l[len(l)//2]
    return l

def minmax(l = []):
    return min(l),max(l)

def searching(l = []):
    num = input("Do you want to add element Y/N:").strip()
    if num.lower() == "y":
        l.append(input("Enter the element you want to append: "))
        searching(l)
    elif num.lower() == "n":
        if "python" in l:
            print("python is present in the list at position: ",l.index("python"))
        elif "Python" in l:
            print("python is present in the list at position: ",l.index("Python"))
        else:
            print("Python is not present in the list")
    else:
        print("You pressed wrong key please try again")
        searching(l)

if __name__ == "__main__":
    while(1):
        num = int(input("*****Menu Driven*****\n1. Separate Even and odd elements\n2. Merge and sort two lists\n3. Updating first element and deleting middle element\n4. Minimum and maximum value of a list\n5. Add names into list and check for python\n6. Exit\n"))
        if num == 1:

```

```

l = list(map(int,input("Enter the element of the list: ").strip().split()))

l1,l2 = even_odd(l)

print(f"Even list: {l1}\nOdd list: {l2}")

elif num == 2:

    l1 = list(map(int,input("Enter the element of first list: ").strip().split()))

    l2 = list(map(int,input("Enter the element of second list: ").strip().split()))

    l = merge_sort(l1,l2)

    print("Merged and sorted list: {}".format(l))

elif num == 3:

    l = list(map(str,input("Enter the element of the list: ").strip().split()))

    l = update_delete(l)

    print(f"Modified list = {l}")

elif num == 4:

    l = list(map(int,input("Enter the element of the list: ").strip().split()))

    mini, maxi = minmax(l)

    print(f"Maximum element of the list: {maxi}\nMinimum element of the list:{mini}")

elif num == 5:

    searching([])

elif num == 6:

    break

else:

    print("you pressed wrong key please try again")

```

## Output:

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python\exp3> python exp3.py
*****Menu Driven*****
1. Separate Even and odd elements
2. Merge and sort two lists
3. Updating first element and deleting middle element
4. Minimum and maximum value of a list
5. Add names into list and check for python
6. Exit
1
Enter the element of the list: 56 89 2 6 546 464 5
Even list: [56, 2, 6, 546, 464]
Odd list: [89, 5]
*****Menu Driven*****
1. Separate Even and odd elements
2. Merge and sort two lists
3. Updating first element and deleting middle element
4. Minimum and maximum value of a list
5. Add names into list and check for python
6. Exit
2
Enter the element of first list: 46 11 78 22 662 45 798 223 69
Enter the element of second list: 12 46 889 2 54
Merged and sorted list: [2, 11, 12, 22, 45, 46, 46, 54, 69, 78, 223, 662, 798, 889]
*****Menu Driven*****
1. Separate Even and odd elements
2. Merge and sort two lists
3. Updating first element and deleting middle element
4. Minimum and maximum value of a list
5. Add names into list and check for python
6. Exit
3
Enter the element of the list: apple mango orange pineapple watermelon
Enter First value to update: grapes
Modified list = ['grapes', 'mango', 'pineapple', 'watermelon']
*****Menu Driven*****
1. Separate Even and odd elements
2. Merge and sort two lists
3. Updating first element and deleting middle element
4. Minimum and maximum value of a list
5. Add names into list and check for python
6. Exit
4
Enter the element of the list: 89 56 1 -999 36 56 45 2
Maximum element of the list: 89
Minimum element of the list: -999
```

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python\exp3> python exp3.py
*****Menu Driven*****
1. Separate Even and odd elements
2. Merge and sort two lists
3. Updating first element and deleting middle element
4. Minimum and maximum value of a list
5. Add names into list and check for python
6. Exit
5
Do you want to add element Y/N: y
Enter the element you want to append: Ruby on rail
Do you want to add element Y/N: Y
Enter the element you want to append: R
Do you want to add element Y/N: Y
Enter the element you want to append: Java
Do you want to add element Y/N: d
You pressed wrong key please try again
Do you want to add element Y/N: Y
Enter the element you want to append: Java script
Do you want to add element Y/N: Y
Enter the element you want to append: Python
Do you want to add element Y/N: n
python is present in the list at position: 4
*****Menu Driven*****
1. Separate Even and odd elements
2. Merge and sort two lists
3. Updating first element and deleting middle element
4. Minimum and maximum value of a list
5. Add names into list and check for python
6. Exit
[]
```

## Tuples:

### Code:

class Student:

```
def __init__(self):
```

```
    self.tup = tuple()
```

```
def addinfo(self):
```

```
    roll, name = int(input("Enter roll no: ")), input("Enter name: ")
```

```
    sub1, sub2, sub3 = map(int, input("Enter Three subjects marks: ").strip().split())
```

```
    l = list(self.tup)
```

```
    l.append((roll, name, sub1, sub2, sub3))
```

```
    self.tup = tuple(l)
```

```
    print("Inserted Tuple:", self.tup[len(self.tup)-1])
```

```

def fetchinfo(self):

    print(*[str(x)+"\n" for x in self.tup if x[1].lower() == "python"])

def display(self):

    print(*[str(x)+"\n" for x in sorted(self.tup,key= lambda x: x[1])])


if __name__ == "__main__":

    stud = Student()

    while True:

        num = int(input("*****Menu Driven*****\n1. To add student information\n2. To fetch
all students where, name == python \n3. To display all student information in sorted order.\n4. Exit\n"))

        if num == 1:

            stud.addinfo()

        elif num == 2:

            stud.fetchinfo()

        elif num == 3:

            stud.display()

        elif num ==4:

            break

        else:

            print("You pressed wrong key please try again")

```

## Output

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python\exp4> python exp4.py
*****Menu Driven*****
1. To add student information
2. To fetch all students where, name == python
3. To display all student information in sorted order.
4. Exit
1
Enter roll no: 55
Enter name: Json
Enter Three subjects marks: 98 95 96
Inserted Tuple: (55, 'Json', 98, 95, 96)
*****Menu Driven*****
1. To add student information
2. To fetch all students where, name == python
3. To display all student information in sorted order.
4. Exit
1
Enter roll no: 15
Enter name: Adnan
Enter Three subjects marks: 55 65 60
Inserted Tuple: (15, 'Adnan', 55, 65, 60)
*****Menu Driven*****
1. To add student information
2. To fetch all students where, name == python
3. To display all student information in sorted order.
4. Exit
1
Enter roll no: 77
Enter name: Python
Enter Three subjects marks: 65 55 45
Inserted Tuple: (77, 'Python', 65, 55, 45)
*****Menu Driven*****
1. To add student information
2. To fetch all students where, name == python
3. To display all student information in sorted order.
4. Exit
1
Enter roll no: 88
Enter name: python
Enter Three subjects marks: 88 95 99
Inserted Tuple: (88, 'python', 88, 95, 99)

*****Menu Driven*****
1. To add student information
2. To fetch all students where, name == python
3. To display all student information in sorted order.
4. Exit
3
(15, 'Adnan', 55, 65, 60)
(55, 'Json', 98, 95, 96)
(77, 'Python', 65, 55, 45)
(88, 'python', 88, 95, 99)

*****Menu Driven*****
1. To add student information
2. To fetch all students where, name == python
3. To display all student information in sorted order.
4. Exit
2
(77, 'Python', 65, 55, 45)
(88, 'python', 88, 95, 99)

*****Menu Driven*****
1. To add student information
2. To fetch all students where, name == python
3. To display all student information in sorted order.
4. Exit
4
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python\exp4>
```

Sets:

Code:

#A accept 2 input string form user

def intake():

str1 =input("Enter first string: ")

str2 =input("Enter second string: ")

print("first string is {} and second string is {}".format(str1, str2))

print(str1, str2)

#B print common letter from input string(set insertion)

def com():

str1 =input("Enter first string: ")

str2 =input("Enter second string: ")

setA=set(str1)

setB=set(str2)

```
common=list(setA&setB)
```

```
print("Common letters in both the set is: ")
```

```
for element in common:
```

```
    print(element)
```

```
#C Display letters which are in the first string but not in the second(set difference)
```

```
def diff():
```

```
    str1 =input("Enter first string: ")
```

```
    str2 =input("Enter second string: ")
```

```
    setA = set(str1)
```

```
    setB = set(str2)
```

```
    difference=list(setA-setB)
```

```
    print("Elements which are in A but not in B :")
```

```
    for element in difference:
```

```
        print(element)
```

```
#D Display set of common letters in both the strings (set union)
```

```
def un():
```

```
    str1 =input("Enter first string: ")
```

```
    str2 =input("Enter second string: ")
```

```
    setA = set(str1)
```

```
    setB = set(str2)
```

```
    union=list(setA|setB)
```

```
    print("Elements in both the sets: ")
```

```
    for element in union:
```

```
        print(element)
```

```
#E Displays letters which are in the two strings but not in both(symmetric difference)
```

```
def uncommon():
```

```
    str1 =input("Enter first string: ")
```

```

str2 =input("Enter second string: ")

setA = set(str1)

setB = set(str2)

uncommon=list(setA^setB)

print("Element present in the strings but not in both the string: ")

for element in uncommon:

    print(element)

if __name__=="__main__":

    while True:

        num = int(input("*****Menu Driven*****\n1. TO input strings \n2.To check common
letters between two strings \n3. To check letters which are in the first string but not in the second.\n4. To
check set of common letters in both the strings \n5. To check letters which are in the both the strings but
not in both \n6. Exit\n"))

        if num == 1:

            intake()

        elif num == 2:

            com()

        elif num == 3:

            diff()

        elif num ==4:

            un()

        elif num==5:

            uncommon()

        elif num==6:

            break

    else:

        print("Wrong key")

```

**Output:**

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp5> python exp-5.py
*****Menu Driven*****
1. TO input strings
2.To check common letters between two strings
3. To check letters which are in the first string but not in the second.
4. To check set of common letters in both the strings
5. To check letters which are in the both the strings but not in both
6. Exit
1
Enter first string: Star Platinum
Enter second string: Za Wurdoo
first string is Star Platinum and second string is Za Wurdoo
Star Platinum Za Wurdoo
*****Menu Driven*****
1. TO input strings
2.To check common letters between two strings
3. To check letters which are in the first string but not in the second.
4. To check set of common letters in both the strings
5. To check letters which are in the both the strings but not in both
6. Exit
2
Enter first string: I, Giorno Giovanna, have a dream
Enter second string: Kono Giorno Giovanna niwa yumegaru
Common letters in both the set is:
r
a
v

o
G
e
i
m
n
```



```
*****Menu Driven*****
```

1. TO input strings
- 2.To check common letters between two strings
3. To check letters which are in the first string but not in the second.
4. To check set of common letters in both the strings
5. To check letters which are in the both the strings but not in both
6. Exit

3

Enter first string: Korega Requiem Desuka

Enter second string: This is Requiem

Elements which are in A but not in B :

D

r

a

o

K

k

g

```
*****Menu Driven*****
```

1. TO input strings
- 2.To check common letters between two strings
3. To check letters which are in the first string but not in the second.
4. To check set of common letters in both the strings
5. To check letters which are in the both the strings but not in both
6. Exit

4

Enter first string: Jonathan Joester

Enter second string: Joseph Joester

Elements in both the sets:

t

o

h

e

p

n

r

a

s

J

```
*****Menu Driven*****
```

1. TO input strings
- 2.To check common letters between two strings
3. To check letters which are in the first string but not in the second.
4. To check set of common letters in both the strings
5. To check letters which are in the both the strings but not in both
6. Exit

5

Enter first string: Dempunki Berserk

Enter second string: Band of Hawk

Element present in the strings but not in both the string:

w

D

r

a

s

u

o

f

e

m

H

p

i

d

```
*****Menu Driven*****
```

1. TO input strings
- 2.To check common letters between two strings
3. To check letters which are in the first string but not in the second.
4. To check set of common letters in both the strings
5. To check letters which are in the both the strings but not in both
6. Exit

6

PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp5>

Dictionary:

**Code:**

```
# Create key/value pair dictionary
```

```
#global dictionary
```

```
my_dict={1: "java", 2: "c", 3: "python"}
```

```
#update dictionary
```

```
def update():
```

```
    key=int(input("enter the key to be updated: "))
```

```
    value=input("Enter value: ")
```

```
    my_dict[key]=value
```

```
    print("updated dictionary: ",my_dict)
```

```
#concatenate dictionary
```

```
def concatenate():
```

```
    key_lst = []
```

```
    value_lst = []
```

```
    n = int(input("Enter number of pairs: "))
```

```
    # iterating till the range
```

```
    for i in range(0, n):
```

```
        keys = int(input("enter key: "))
```

```
        key_lst.append(keys)
```

```
        value = (input("enter value: "))
```

```
        value_lst.append(value)
```

```
my_dict_2= dict(zip(key_lst, value_lst))
```

```
my_dict.update(my_dict_2)
```

```
print("after concatenation:= ", my_dict)
```

```
return my_dict
```

```
def delete():
```

```
    #delete value from dictionary
```

```
    key=int(input("enter the key to be deleted:="))
```

```
    del my_dict[key]
```

```
    print("Dictionary after deleting key {} associated with its value is:= {}".format(key, my_dict))
```

```
def find():
```

```
    #find a key and print its value
```

```
    key_list=list(my_dict.keys())
```

```
    value_list=list(my_dict.values())
```

```
    print("keys from dictionary my_dict",key_list)
```

```
    x=int(input("Enter the key to be viewed:="))
```

```
    value=key_list.index(x)
```

```
    print("associated value to the selected key {} is : {}".format(x, value_list[value]))
```

```
def combine_list():
```

```
    key_list = []
```

```
    n = int(input("Enter number of elements : "))
```

```
    # iterating till the range
```

```
    for i in range(0, n):
```

```
        element1 = int(input("enter key list:"))
```

```
        key_list.append(element1)
```

```
    value_list = []
```

```
    # iterating till the range
```

```
    for i in range(0, n):
```

```
        value= (input("enter value list: "))
```

```

    value_lst.append(value)

#map two lists into dictionary
combined_list=dict(zip(key_lst, value_lst))

print("after combining two list of keys(key_list) and values(value_list)", combined_list)

if __name__=="__main__":

    print(my_dict)

    while True:

        num = int(input("*****Menu Driven*****\n1. TO update dicitonary \n2.To
concatenate dictionary \n3. To delete values from dictionary.\n4. To find value from dictionary \n5. To
create dictionary from lists. \n6. Exit\n"))

        if num == 1:

            update()

        elif num == 2:

            concatenate()

        elif num == 3:

            delete()

        elif num ==4:

            find()

        elif num==5:

            combine_linst()

        elif num==6:

            break

        else:

            print("Wrong key")

```

## Output:

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python\exp6> python exp-6.py
{1: 'java', 2: 'c', 3: 'python'}
*****Menu Driven*****
1. TO update dicitonary
2.To concatenate dictionnary
3. To delete values from dictionary.
4. To find value from dictionary
5. To create dictionary from lists.
6. Exit
1
enter the key to be updated: 1
Enter value: Java Script
updated dictionary: {1: 'Java Script', 2: 'c', 3: 'python'}
*****Menu Driven*****
1. TO update dicitonary
2.To concatenate dictionnary
3. To delete values from dictionary.
4. To find value from dictionary
5. To create dictionary from lists.
6. Exit
2
Enter number of pairs: 3
enter key: 4
enter value: Java
enter key: 5
enter value: R
enter key: 6
enter value: C#
after concatenation:= {1: 'Java Script', 2: 'c', 3: 'python', 4: 'Java', 5: 'R', 6: 'C#'}
*****Menu Driven*****
1. TO update dicitonary
2.To concatenate dictionnary
3. To delete values from dictionary.
4. To find value from dictionary
5. To create dictionary from lists.
6. Exit
3
enter the key to be deleted:=2
Dictionary after deleting key 2 associated with its value is:= {1: 'Java Script', 3: 'python', 4: 'Java', 5: 'R', 6: 'C#'}

*****Menu Driven*****
1. TO update dicitonary
2.To concatenate dictionnary
3. To delete values from dictionary.
4. To find value from dictionary
5. To create dictionary from lists.
6. Exit
4
keys from dictionary my_dict [1, 3, 4, 5, 6]
Enter the key to be viewed:=4
associated value to the selected key 4 is : Java

*****Menu Driven*****
1. TO update dicitonary
2.To concatenate dictionnary
3. To delete values from dictionary.
4. To find value from dictionary
5. To create dictionary from lists.
6. Exit
5
Enter number of elements : 4
enter key list:1
enter key list:2
enter key list:9
enter key list:6
enter value list: JSON
enter value list: REACT
enter value list: Angular JS
enter value list: Vue Js
after combining two list of keys(key_list) and values(value_list) {1: 'JSON', 2: 'REACT', 9: 'Angular JS', 6: 'Vue Js'}
*****Menu Driven*****
1. TO update dicitonary
2.To concatenate dictionnary
3. To delete values from dictionary.
4. To find value from dictionary
5. To create dictionary from lists.
6. Exit
6
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python\exp6> 
```

## EXPERIMENT NO. 2

**Aim :** Demonstrate exception handling and inheritance.

**Resources Required:** Consumables – Printer Pages for printouts.

### Theory:

Exceptional Handling:

Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them:

- a. Exception Handling
- b. Assertions

List of Standard Exceptions –

Sr.No	Exception Name	Description
1	Exception	Base class for all exceptions
2	StandardError	Base class for all built-in exceptions except StopIteration and SystemExit.
3	ArithmeticError	Base class for all errors that occur for numeric calculation.
4	OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
5	NameError	Raised when an identifier is not found in the local or global namespace.
6	IOError	Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
7	TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.
8	ValueError	It is raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.

Classes, Objects and Inheritance:

### Creating Classes:

The class statement creates a new class definition. The name of the class immediately follows the keyword class followed by a colon as follows –

```
class ClassName:  
'Optional class documentation string' class_suite
```

- The class has a documentation string, which can be accessed via `ClassName.doc`.
- The `class_suite` consists of all the component statements defining class members, data attributes and functions.

### Creating Instance Objects:

To create instances of a class, you call the class using class name and pass in whatever arguments its `__init__` method accepts.

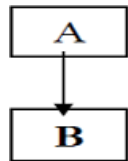
### Accessing Attributes

You access the object's attributes using the dot operator with object.

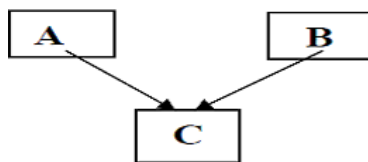
Inheritance means deriving properties from other classes.

### Types of Inheritance:

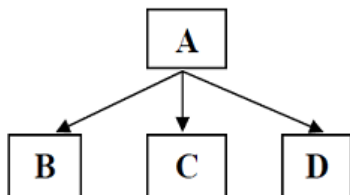
1. **Single Inheritance:** A derived class with only one base class is called as Single Inheritance.



2. **Multiple Inheritances:** A derived class with several base classes is called Multiple Inheritance.

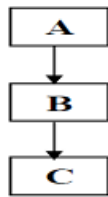


3. **Hierarchical Inheritance:** More than one class may inherit the features of one class this process is called as Hierarchical Inheritance.

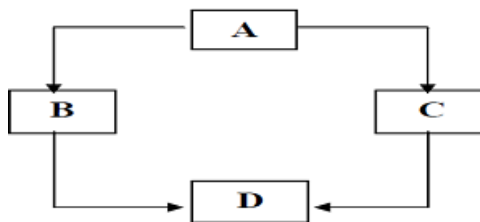


4. **Multilevel Inheritance:** The mechanism of deriving a class from another derived class is

called Multilevel Inheritance.



5. Hybrid Inheritance: There could be situations where we need to apply one or more types of inheritances to design a program this process is called Hybrid Inheritance.



Method overloading: In Python you can define a method in such a way that there are multiple ways to call it. Given a single method or function, we can specify the number of parameters ourselves. Depending on the function definition, it can be called with zero, one, two or more parameters. This is known as method overloading.

Example: class operation:

```
def abc(self, a=None):  
    if a is not None:  
        print ('first num is' + a)  
    else:  
        print ('no number assigned ') obj = operation()  
        obj.abc () obj.abc('10')
```

Method overriding: Overriding is the ability of a class to change the implementation of a method provided by one of its ancestors. In Python method overriding occurs simply defining in the child class a method with the same name of a method in the parent class.

Example

```
class Parent(object):  
    def __init__(self):  
        self.value = 5  
    def get_value(self): ### overridden method return self.value  
class Child(Parent):  
    def get_value(self):  
return self.value + 1
```

**Conclusion: Hence, We have successfully demonstrate the concept of exception handling and Inheritance.**



Exceptional Handling:

**Code:**

A:

```
import math

import warnings

warnings.filterwarnings("ignore", category=DeprecationWarning)

number_list = [6,-9,420.69,'banana']

for number in number_list:

    try:

        number_factorial = math.factorial(number)

    except TypeError:

        print("Factorial is not supported for given input type. \n")

    except ValueError:

        print(f"Factorial only accepts whole values. {number} is not a whole value. \n")

    else:

        print(f"The factorial of {number} is {number_factorial}\n")

    finally:

        print("Release any resources in use. \n")
```

## Output:

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp9> python exceptionhandling.py
The factorial of 6 is 720

Release any resources in use.

Factorial only accepts whole values. -9 is not a whole value.

Release any resources in use.

Factorial only accepts whole values. 420.69 is not a whole value.

Release any resources in use.

Factorial is not supported for given input type.

Release any resources in use.
```

B:

#numberGuessing

```
class ValueTooSmallError(Exception):
```

```
    #Raised when the input value is too small
```

```
    pass
```

```
class ValueTooLargeError(Exception):
```

```
    #Raised when the input value is too large
```

```
    pass
```

```
number = 10
```

```
while True:
```

```
    try:
```

```
        i_num = int(input("Guess the number: "))
```

```
        if i_num < number:
```

```
            raise ValueTooSmallError
```

```
        elif i_num > number:
```

```
            raise ValueTooLargeError
```

```
        break
```

```
    except ValueTooSmallError:
```

```

    print("This value is too small, try again!")

    print()

except ValueError:

    print("This value is too large, try again!")

    print()

print("Congratulations! You guessed it correctly.")

```

#### Output:

```

PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp9> python userdefinedexception.py
Guess the number: 56
This value is too large, try again!

Guess the number: 99
This value is too large, try again!

Guess the number: 69
This value is too large, try again!

Guess the number: 7
This value is too small, try again!

Guess the number: 9
This value is too small, try again!

Guess the number: 11
This value is too large, try again!

Guess the number: 10
Congratulations! You guessed it correctly.

```

Inheritance:

#### Single Inheritance:

Code:

```

class Animal:
    def __init__(self):
        self.kingdomname = "Animalia"
    def kingdom(self):
        print("Kingdom: ",self.kingdomname)

class Cow(Animal):
    def __init__(self):
        super().__init__()
        self.name = "Cow"
        self.voc = "Mooooo"

```

```

def naam(self):
    print("Name: ",self.name)

def voice(self):
    print(f"{self.name} {self.voc}")

if __name__ == "__main__":
    cow1 = Cow()
    cow1.kingdom()
    cow1.naam()
    cow1.voice()

```

Output:

```

PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> python sinheritane.py
Kingdom: Animalia
Name: Cow
Cow Mo000
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> 

```

### Multiple Inheritance:

Code:

```

class Father:
    fathername = ""

    def show_father(self):
        print(self.fathername)

class Mother:
    mothername = ""

    def show_mother(self):
        print(self.mothername)

class Son(Father, Mother):
    def show_parent(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)

if __name__ == "__main__":

s1 = Son() # Object of Son class
s1.fathername = "Joseph Joester"
s1.mothername = "Lily"
s1.show_parent()

```

Output:

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> python Minheritance.py
Father : Joseph Joester
Mother : Lily
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> █
```

### Hierarchical Inheritance:

Code:

```
class Parent:
    parentname = ""
    childname = ""

    def show_parent(self):
        print("Parent Name: ",self.parentname)

class Son(Parent):
    def show_child(self):
        print("Son Name: ",self.childname)

class Daughter(Parent):
    def show_child(self):
        print("Daughter Name: ",self.childname)

if __name__ == "__main__":
    s1 = Son()
    s1.parentname = "Cujoh"
    s1.childname = "Jotaro"
    s1.show_parent()
    s1.show_child()

    d1 = Daughter()
    d1.childname = "Jolyne"
    d1.parentname = "Jotaro"
    d1.show_parent()
    d1.show_child()
```

Output:

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> python hierarchical.py
Parent Name: Cujoh
Son Name: Jotaro
Parent Name: Jotaro
Daughter Name: Jolyne
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> █
```

### Multilevel Inheritance:

Code:

```

class Family:
    def show_family(self):
        print("This is Cujoh Family:")
class Father(Family):
    fathername = ""
    def show_father(self):
        print(self.fathername)
class Mother(Family):
    mothername = ""

    def show_mother(self):
        print(self.mothername)

class Child(Father, Mother):
    def __init__(self):
        super().__init__()
        self.childname = ""
    def show_parent(self):
        print("Son name: ",self.childname)
        print("Father :", self.fathername)
        print("Mother :", self.mothername)

if __name__ == "__main__":
    s1 = Child()
    s1.childname = "Jotaro Cujoh"
    s1.fathername = "Sadao Cujoh"
    s1.mothername = "Holy Cujoh"
    s1.show_family()
    s1.show_parent()

```

Output:

```

PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> python multilevel.py
This is Cujoh Family:
Son name: Jotaro Cujoh
Father : Sadao Cujoh
Mother : Holy Cujoh
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> 

```

## EXPERIMENT NO. 3

**Aim:** Exploring files and directories.

- A. Python program to read the content of file and write in another file.
- B. Program to append data to existing file and then display entire file.
- C. Program to count number of lines, words and characters in a file.
- D. Program to display file available in current directory.

**Resources Required:** Consumables – Printer Pages for printouts.

### Theory:

Data is very important. Every organization depends on its data for continuing its business operations. To store the data in computer we need files. There are certain advantages of storing data in file:

- a. to store huge amount of data When data is stored in a file, it is stored permanently.
- b. It is possible to update the file data.
- c. Files are highly useful.

In Python, there are two types of files. They are

- 1. Text File-: store the data in the form of characters
- 2. Binary File-: store the entire data in the form of bytes.

It is very important to know how to create files, store data in the files and retrieve the data from the files in python. To do any operations on files, first of all we should open the files.

Opening a File in order to open a file for writing or use in Python, you must rely on the built-in open () function.

As explained above, open ( ) will return a file object, so it is most commonly used with two arguments. An argument is nothing more than a value that has been provided to a function, which is relayed when you call it. So, for instance, if we declare the name of a file as “Test File,” that name would be considered an argument. The syntax to open a file object in Python is:

file\_object = open(“filename”, “mode”) where file\_object is the variable to add the file object. Mode

Including a mode argument is optional because a default value of ‘r’ will be assumed if it is omitted. The ‘r’ value stands for read mode, which is just one of many. The modes are:

- ‘r’ – Read mode which is used when the file is only being read
- ‘w’ – Write mode which is used to edit and write new information to the file (any existing files with the same name will be erased when this mode is activated)
- ‘a’ – Appending mode, which is used to add new data to the end of the file; that is new information is automatically amended to the end
- ‘r+’ – Special read and write mode, which is used to handle both actions when working with a file.

So, let’s take a look at a quick example. f=open(“workfile”, “w”)

Print f

This snippet opens the file named “workfile” in writing mode so that we can make changes to it. Just create the file named “testfile.txt” and leave it blank. If you need to extract a string that contains all characters in the file, then

```
file=open(“testfile.txt”, “r”)
```

```
print file.read()
```

Using the File Write Method the file write method is that it only requires a single parameter, which is the

string you want to be written. This method is used to add information or content to an existing file. To start a new line after you write data to the file, you can add an EOL character. Closing a File when you're done working, you can use the `fh.close()` command to end things.

**Conclusion:** Hence, we have successfully study the concept of file handling in python.

A:

**Code:**

```
file1 = open("dio.txt","w")
file1.write("KONO DIO DAAA")
file1.close()
file2 = open("wurdo.txt","w")
file2.write("You expected Output but it was me DIO")
file2.close()
file1 = open("dio.txt","r")
file2 = open("wurdo.txt","r")
file3 = open("output.txt","w")
file3.write(file1.read())
file3.write(file2.read())
file1.close()
file2.close()
file2.close()
print("Data copied successfully!!!")
```

**Output:**

```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp10\copyfiles.py"
Data copied successfully!!!
[Done] exited with code=0 in 0.13 seconds
```

**Content of files:**

DIO.txt: KONO DIO DAAA  
wurdo.txt: You expected Output but it was me DIO  
output.txt: KONO DIO DAAAYou expected Output but it was me DIO

B:

**Code:**

```
f=open("Adnan.txt","a+")
f.write(" Giorno Giovanna")
print("Data Appended Sucessfully")
f.close()
```

**Output:**

```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp10\second\append file.py"
Data Appended Sucessfully
[Done] exited with code=0 in 0.08 seconds
```



### Content of a file:

Before appending: Adnan Shaikh

After appending: Adnan Shaikh   Giorno Giovanna

C:

#### Code:

```
file = open("Count.txt", "r")
```

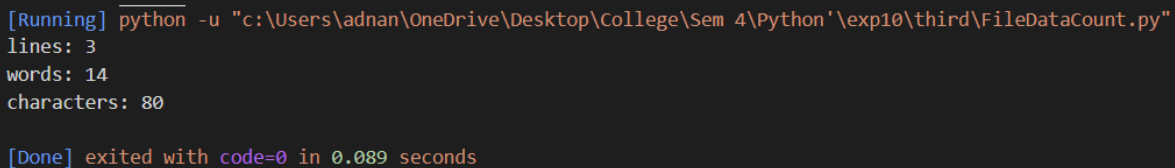
```
number_of_lines = 0  
number_of_words = 0  
number_of_characters = 0  
for line in file:  
    line = line.strip("\n")
```

```
words = line.split()  
number_of_lines += 1  
number_of_words += len(words)  
number_of_characters += len(line)
```

```
file.close()
```

```
print("lines:", number_of_lines)  
print("words:", number_of_words)  
print("characters:", number_of_characters)
```

Output:



```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp10\third\FileDataCount.py"  
lines: 3  
words: 14  
characters: 80  
  
[Done] exited with code=0 in 0.089 seconds
```

### Content of a file:

Hello, this is Adnan

Welcome to my wooooooooorld

Ok, understandable have a great day.

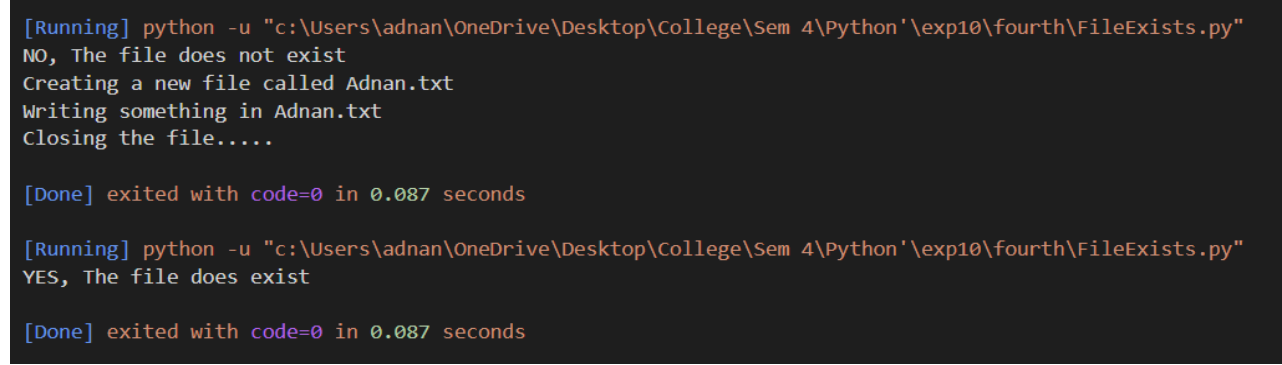
D:

#### Code:

```
import os  
from pathlib import Path  
if os.path.exists("Adnan.txt"):  
    print("YES, The file does exist")  
else:  
    print("NO, The file does not exist\nCreating a new file called Adnan.txt")
```

```
f1 = open("Adnan.txt","w+")
print("Writing something in Adnan.txt")
f1.write("Well well what do we have we here")
print("Closing the file.....")
f1.close()
```

#### Output:



```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp10\fourth\FileExists.py"
NO, The file does not exist
Creating a new file called Adnan.txt
Writing something in Adnan.txt
Closing the file.....

[Done] exited with code=0 in 0.087 seconds

[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp10\fourth\FileExists.py"
YES, The file does exist

[Done] exited with code=0 in 0.087 seconds
```

**Content of a file:** Well well what do we have we here

## EXPERIMENT NO. 4

**Aim:** To study GUI designing using Tkinter in Python.

**Resources Required:** Consumables – Printer Pages for printouts.

### Theory:

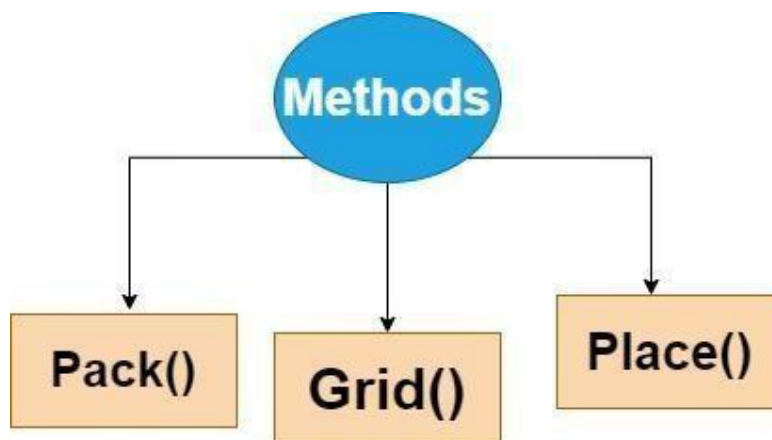
#### Tkinter:

Tkinter is the most commonly used library for developing GUI (Graphical User Interface) in Python. It is a standard Python interface to the Tk GUI toolkit shipped with Python. As Tk and Tkinter are available on most of the Unix platforms as well as on the Windows system, developing GUI applications with Tkinter becomes the fastest and easiest.

In order to organize or arrange or place all the widgets in the parent window, Tkinter provides us the geometric configuration of the widgets. The GUI Application Layout is mainly controlled by Geometric Managers of Tkinter.

It is important to note here that each window and Frame in your application is allowed to use only one geometry manager. Also, different frames can use different geometry managers, even if they're already assigned to a frame or window using another geometry manager.

There are mainly three methods in Geometry Managers:



Let us discuss each method in detail one by one.

#### 1. Tkinter pack() Geometry Manager

The pack() method mainly uses a packing algorithm in order to place widgets in a Frame or window in a specified order.

Algorithm:

The steps of Packing algorithm are as follows:

- Firstly this algorithm will compute a rectangular area known as a Parcel which is tall (or wide) enough to hold the widget and then it will fill the remaining width (or height) in the window with blank space.
- It will center the widget until any different location is specified.

This method is powerful but it is difficult to visualize.

The possible options as a parameter to this method are given below:

- fill

The default value of this option is set to NONE. Also, we can set it to X or Y in order to determine whether the widget contains any extra space.

- side

This option specifies which side to pack the widget against. If you want to pack widgets vertically, use TOP which is the default value. If you want to pack widgets horizontally, use LEFT.

- expand

This option is used to specify whether the widgets should be expanded to fill any extra space in the geometry master or not. Its default value is false. If it is false then the widget is not expanded otherwise widget expands to fill extra space.

### Tkinter pack() with Parameters

Let's take a few more code examples using the parameters of this function like fill, side and, expand.

You can set the fill argument in order to specify in which direction you want the frames should fill. If you want to fill in the horizontal direction then the option is tk.X, whereas, tk.Y is used to fill vertically, and to fill in both directions tk.BOTH is used.

## 2. Tkinter grid() Geometry Manager

The most used geometry manager is grid() because it provides all the power of pack() function but in an easier and maintainable way.

The grid() geometry manager is mainly used to split either a window or frame into rows and columns.

You can easily specify the location of a widget just by calling `grid()` function and passing the row and column indices to the `row` and `column` keyword arguments, respectively.

Index of both the row and column starts from 0, so a row index of 2 and a column index of 2 tells the `grid()` function to place a widget in the third column of the third row (0 is first, 1 is second and 2 means third).

Here is the syntax of the `grid()` function:

The possible options as a parameter to this method are given below:

- Column

This option specifies the column number in which the widget is to be placed. The index of leftmost column is 0.

- Row

This option specifies the row number in which the widget is to be placed. The topmost row is represented by 0.

- Columnspan

This option specifies the width of the widget. It mainly represents the number of columns up to which, the column is expanded.

- Rowspan

This option specifies the height of the widget. It mainly represents the number of rows up to which, the row is expanded.

- `padx, pady`

This option mainly represents the number of pixels of padding to be added to the widget just outside the widget's border.

- `ipadx, ipady`

This option is mainly used to represent the number of pixels of padding to be added to the widget inside the widget's border.

- Sticky

If any cell is larger than a widget, then `sticky` is mainly used to specify the position of the widget inside the cell. It is basically concatenation of the sticky letters which represents the position of the widget. It may be N, E, W, S, NE, NW, NS, EW, ES

As you can see in the code example above, we have used the `padx` and `pady` parameters because of which padding is applied outside the widget. To add padding inside the Frame widget, use the parameters `ipadx` and `ipady` in your code.

Similarly, do try using other parameters too for the `grid()` geometry manager.

### 3. Trinket `place()` Geometry Manager

The `place()` Geometry Manager organizes the widgets to place them in a specific position as directed by the programmer.

- This method basically organizes the widget in accordance with its `x` and `y` coordinates. Both `x` and `y` coordinates are in pixels.

Thus the origin (where `x` and `y` are both 0) is the top-left corner of the Frame or the window.

Thus, the `y` argument specifies the number of pixels of space from the top of the window, to place the widget, and the `x` argument specifies the number of pixels from the left of the window.

The possible options as a parameter to this method are given below:

- `x, y`

This option indicates the **horizontal and vertical** offset in the pixels.

- `height, width`

This option indicates the **height and weight** of the widget in the pixels.

- `Anchor`

This option mainly represents the **exact position** of the widget within the container. The default value (direction) is **NW** that is (the upper left corner).

- `bordermode`

This option indicates the default value of the border type which is INSIDE and it also refers to ignore the parent's inside the border. The other option is OUTSIDE.

- relx, rely

This option is used to represent the float between 0.0 and 1.0 and it is the **offset in the horizontal and vertical direction**.

- relheight, relwidth

This option is used to represent the float value between 0.0 and 1.0 indicating the fraction of the **parent's height and width**.

**Conclusion:** Hence, We have successfully study the concept of GUI in python.

**Code:**

1) WAP to draw the house on an canvas using

```
Tkinter. from tkinter import *
```

```
root = Tk()
```

```
C = Canvas(root, height = 300, width = 300)
```

```
line = C.create_rectangle(100, 100, 200, 200)
```

```
triangle = C.create_polygon([100, 100, 150, 40, 200, 100], fill="white", outline='black')
```

```
rectangle = C.create_rectangle(130, 140, 170, 200)
```

```
rectangle2 = C.create_rectangle(110, 110, 135, 130)
```

```
rectangle3 = C.create_rectangle(165, 110, 190, 130)
```

```
circle = C.create_oval(135, 60, 165, 90)
```

```
C.pack() root.mainloop()
```

## OUTPUT:



## Code:

Program to implement Calculator to perform different arithmetic operations.

```
from tkinter import *  
  
win = Tk() win.geometry("312x324") win.resizable(0, 0) win.title("Calculator")  
  
def btn_click(item): global expression  
expression = expression + str(item) input_text.set(expression)  
  
def bt_clear(): global expression  
expression = "" input_text.set("")
```



```

def bt_equal(): global expression

result = str(eval(expression)) input_text.set(result) expression = ""

expression = ""

input_text = StringVar()

input_frame = Frame(win, width=312, height=50, bd=0, highlightbackground="black",
highlightcolor="black", highlightthickness=2)

input_frame.pack(side=TOP)

input_field = Entry(input_frame, font=('arial', 18, 'bold'), textvariable=input_text, width=50, bg="#eee",
bd=0, justify=RIGHT)

input_field.grid(row=0, column=0) input_field.pack(ipady=10)

btns_frame = Frame(win, width=312, height=272.5, bg="grey") btns_frame.pack()

# first row

clear = Button(btns_frame, text = "C", fg = "black", width = 32, height = 3, bd = 0, bg = "#eee", cursor =
"hand2", command = lambda: bt_clear()).grid(row = 0, column = 0, columnspan = 3, padx = 1, pady = 1)

divide = Button(btns_frame, text = "/", fg = "black", width = 10, height = 3, bd = 0, bg = "#eee", cursor =
"hand2", command = lambda: btn_click("/")).grid(row = 0, column= 3, padx = 1, pady= 1)

# second row

seven = Button(btns_frame, text = "7", fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor =
"hand2", command = lambda: btn_click(7)).grid(row = 1, column = 0, padx = 1, pady = 1)

eight = Button(btns_frame, text = "8", fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor =
"hand2", command = lambda: btn_click(8)).grid(row = 1, column = 1, padx = 1, pady = 1)

```

```

nine = Button(btns_frame, text = "9", fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor =
"hand2", command = lambda: btn_click(9)).grid(row = 1, column = 2, padx = 1, pady = 1)

multiply = Button(btns_frame, text = "*", fg = "black", width = 10, height = 3, bd = 0, bg = "#eee", cursor =
"hand2", command = lambda: btn_click("*")).grid(row = 1, column = 3, padx = 1, pady = 1)

# third row

four = Button(btns_frame, text = "4", fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor =
"hand2", command = lambda: btn_click(4)).grid(row = 2, column = 0, padx = 1, pady = 1)

five = Button(btns_frame, text = "5", fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor =
"hand2", command = lambda: btn_click(5)).grid(row = 2, column = 1, padx = 1, pady = 1)

six = Button(btns_frame, text = "6", fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor = "hand2",
command = lambda: btn_click(6)).grid(row = 2, column = 2, padx = 1, pady = 1)

minus = Button(btns_frame, text = "-", fg = "black", width = 10, height = 3, bd = 0, bg = "#eee", cursor =
"hand2", command = lambda: btn_click("-")).grid(row = 2, column = 3, padx = 1, pady = 1)

# fourth row

one = Button(btns_frame, text = "1", fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor =
"hand2", command = lambda: btn_click(1)).grid(row = 3, column = 0, padx = 1, pady = 1)

two = Button(btns_frame, text = "2", fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor =
"hand2", command = lambda: btn_click(2)).grid(row = 3, column = 1, padx = 1, pady = 1)

three = Button(btns_frame, text = "3", fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor =
"hand2", command = lambda: btn_click(3)).grid(row = 3, column = 2, padx = 1, pady = 1)

plus = Button(btns_frame, text = "+", fg = "black", width = 10, height = 3, bd = 0, bg = "#eee", cursor =
"hand2", command = lambda: btn_click("+")).grid(row = 3, column = 3, padx = 1, pady = 1)

# fourth row

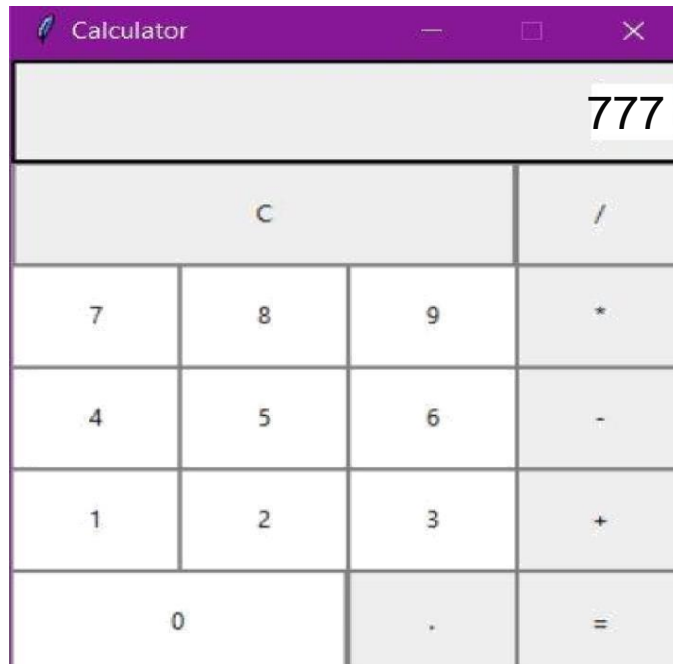
zero = Button(btns_frame, text = "0", fg = "black", width = 21, height = 3, bd = 0, bg = "#fff", cursor =
"hand2", command = lambda: btn_click(0)).grid(row = 4, column = 0, columnspan = 2, padx = 1, pady = 1)

point = Button(btns_frame, text = ".", fg = "black", width = 10, height = 3, bd = 0, bg = "#eee", cursor =
"hand2", command = lambda: btn_click(".")).grid(row = 4, column = 2, padx = 1, pady = 1)

equals = Button(btns_frame, text = "=", fg = "black", width = 10, height = 3, bd = 0, bg = "#eee", cursor =
"hand2", command = lambda: bt_equal()).grid(row = 4, column = 3, padx = 1, pady = 1) win.mainloop()

```

Output:



## EXPERIMENT NO. 5

**Aim:** Menu driven program for data structure using built in function for stack and queue.

**Resources Required:** Consumables – Printer Pages for printouts.

### Theory:

#### Stack

It is a sequence of items that are accessible at only one end of the sequence. Think of a stack as a collection of items that are piled one on top of the other, with access limited to the topmost item. A stack inserts item on the top of the stack and removes item from the top of the stack. It has LIFO (last-in / first-out) ordering for the items on the stack. Also, the inbuilt functions in Python make the code short and simple. To add an item to the top of the list, i.e., to push an item, we use append() function and to pop out an element we use pop() function. These functions work quite efficiently and fast in end operations.

#### Queue

A queue is a sequential storage structure that permits access only at the two ends of the sequence.

We refer to the ends of the sequence as the front and rear. A queue inserts new elements at the rear and removes elements from the front of the sequence. You will note that a queue removes elements in the same order in which they were stored, and hence a queue provides FIFO (first-in / first-out), or FCFS (first-come / first-served), ordering. Implementing queue is a bit different. Time plays an important factor here. During the implementation of stack we used append() and pop() function which was efficient and fast because we inserted and popped elements from the end of the list, but in queue when insertion and pops are made from the beginning of the list, it is slow. This occurs due to the properties of list, which is fast at the end operations but slow at the beginning operations, as all other elements have to be shifted one by one. So, it's preferable to use collections.deque over list, which was specially designed to have fast appends and pops from both the front and back end.

**Conclusion:** Hence, we have successfully written menu driven program for data structure like Stack and Queue in python.

### Code:

Package Content:

```
class Stack:
```

```
    def __init__(self):
        self.stack = []
```

```
    def push(self, val):
```

```

        self.stack.append(val)

def pop(self):
    if len(self.stack) > 0:
        print(self.stack.pop())
    else:
        print("Stack is empty!!!")

def display(self):
    print(self.stack)

class Queue:
    def __init__(self):
        self.queue = []

    def enqueue(self, val):
        self.queue.append(val)

    def dequeue(self):
        if len(self.queue) > 0:
            print(self.queue.pop(0))
        else:
            print("Queue is empty!!!")

    def display(self):
        print(self.queue)

Implementing Package:

from StackQueue.stack_queue import Stack, Queue

if __name__ == "__main__":

    while True:
        num = int(input("*****Menu Driven*****\n1.Implementing
Stack\n2.Implementing Queue\n3.Exit\n"))
        if num == 1:
            s = Stack()
            while True:
                n = int(input("1.Push\n2.Pop\n3.Display\n4.Back\n"))
                if n == 1:
                    s.push(input("Enter element you want to push: "))
                elif n == 2:
                    s.pop()
                elif n == 3:
                    s.display()
                elif n == 4:
                    break
                else:
                    print("You pressed wrong key please try agin")
            elif num == 2:
                q = Queue()
                while True:
                    n = int(input("1.Enqueue\n2.Dequeue\n3.Display\n4.Back\n"))
                    if n == 1:

```

```

        q.enqueue(input("Enter element you want to enqueue: "))
    elif n == 2:
        q.dequeue()
    elif n == 3:
        q.display()
    elif n == 4:
        break
    else:
        print("You pressed wrong key please try agin")
elif num == 3:
    break
else:
    print("You pressed wrong key please try again")

```

### Output:

```

PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python\exp11> python exp11.py
*****Menu Driven*****
1.Implementing Stack
2.Implementing Queue
3.Exit
1
1.Push
2.Pop
3.Display
4.Back
1
Enter element you want to push: Hello
1.Push
2.Pop
3.Display
4.Back
1
Enter element you want to push: World
1.Push
2.Pop
3.Display
4.Back
3
['Hello', 'World']
1.Push
2.Pop
3.Display
4.Back
2
World
1.Push
2.Pop
3.Display
4.Back
3
['Hello']

```

```

1.Push
2.Pop
3.Display
4.Back
2
Hello
1.Push
2.Pop
3.Display
4.Back
2
Stack is empty!!!
1.Push
2.Pop
3.Display
4.Back
4

```

```

*****Menu Driven*****
1.Implementing Stack
2.Implementing Queue
3.Exit
2
1.Enqueue
2.Dequeue
3.Display
4.Back
1
Enter element you want to enqueue: KONO
1.Enqueue
2.Dequeue
3.Display
4.Back
1
Enter element you want to enqueue: DIO
1.Enqueue
2.Dequeue
3.Display
4.Back
1
Enter element you want to enqueue: DA
1.Enqueue
2.Dequeue
3.Display
4.Back
3
['KONO', 'DIO', 'DA']
1.Enqueue
2.Dequeue
3.Display
4.Back
2
KONO

```

```

1.Enqueue
2.Dequeue
3.Display
4.Back
2
DIO
1.Enqueue
2.Dequeue
3.Display
4.Back
3
['DA']
1.Enqueue
2.Dequeue
3.Display
4.Back
2
DA
1.Enqueue
2.Dequeue
3.Display
4.Back
2
Queue is empty!!!
1.Enqueue
2.Dequeue
3.Display
4.Back
4
*****Menu Driven*****
1.Implementing Stack
2.Implementing Queue
3.Exit
3
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python\exp11>

```

## EXPERIMENT NO. 6

**Aim:** Program to demonstrate CRUD (create, read, update and delete) operations on database (SQLite/ MySQL) using python

**Resources Required:** Consumables – Printer Pages for printouts.

### Theory:

Step1: Install python and set the path.

Step2: Using any IDE or in terminal write the following command to install mysql connector:

```
pip install mysql-connector-python
```

or

```
Python -m pip install mysql-connector-python (if upper command doesn't work)
```

After installing above library we're ready to use it in our python files

(Note: If you create Virtual Environment you need to install it in your Virtual Environment.)

Step3: Create a Python file of your desired name(extension: .py)

Step4: You need to write following code at the top to import mysql connector:

```
from mysql import connector
```

Step5: Now we need to connect to database using mysql.connector object, for that we have following syntax:

```
Object_name = connector.connect( host="hostname", username="db_username", password =  
"user_password", database = "db_name")
```

Hostname: In our case hostname is local host since we're not hosting it anywhere

db\_username: It is a database username it should exist or else code will raise an error.

Password: user password should be correct or code will raise an error.

Db\_name: database name which you're going to use if not exist it will raise an error.

Step6: If everything is correct in Step5 we have successfully connected to our database and we can write queries using our Object.

**Conclusion:** We have successfully demonstrate MySQL database connectivity in python.

### Code:

```
from mysql import connector
```



```

mydb = connector.connect(
    host = "localhost",
    username = "root",
    password = "admin",
    database = "pythontemp"
)

mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE IF NOT EXISTS customers(customer_id INT
AUTO_INCREMENT PRIMARY KEY,name VARCHAR(30))")

mycursor.execute("SHOW TABLES")

print(*[x for x in mycursor])

try:
    mycursor.execute("ALTER TABLE customers ADD address VARCHAR(255)")
except:
    print("Column already exist")

mycursor.execute("DESC customers")

print(*[x for x in mycursor])

sql = "INSERT INTO customers(name,address) values(%s,%s)"

val = [
    ("Levi Ackermann", "Wall Rose"),
    ("Eren Jaeger", "Wall Maria"),
    ("Lalatinna", "Konosuba"),
    ("Kaneki Kun", "Re"),
    ("Rias Gremory", "DxD"),
]

mycursor.executemany(sql,val)

mydb.commit()

print(mycursor.rowcount," was inserted")

mycursor.execute("SELECT * FROM customers ORDER BY name")

result = mycursor.fetchall()

for x in result:
    print(x)

```

Output:

```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\DBMS\Practicals\Practical-10\dbconnect.py"
('customers',)
('customer_id', b'int', 'NO', 'PRI', None, 'auto_increment') ('name', b'varchar(30)', 'YES', '', None, '') ('address', b'varchar(255)', 'YES', '', None, '')
5 was inserted
(2, 'Eren Jaeger', 'Wall Maria')
(4, 'Kaneki Kun', 'Re')
(3, 'Lalatinna', 'Konosuba')
(1, 'Levi Ackermann', 'Wall Rose')
(5, 'Rias Gremory', 'DxD')

[Done] exited with code=0 in 0.261 seconds
```

## EXPERIMENT NO. 7

**Aim:** Creation of simple socket for basic information exchange between server and client

**Resources Required:** Consumables – Printer Pages for printouts.

### Theory:

Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents. Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on. The socket library provides specific classes for handling the common transports as well as a generic interface for handling the rest. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server. They are the real backbones behind web browsing. In simpler terms there is a server and a client. Socket programming is started by importing the socket library and making a simple socket.

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Here we made a socket instance and passed it two parameters. The first parameter is AF\_INET and the second one is SOCK\_STREAM. AF\_INET refers to the address family ipv4. The SOCK\_STREAM means connection oriented TCP protocol. Now we can connect to a server using this socket.

### Connecting to a server:

Note that if any error occurs during the creation of a socket then a socket.error is thrown and we can only connect to a server by knowing its IP.

You can find the ip of the server by using this: \$ ping [www.google.com](http://www.google.com)

You can also find the ip using python: `import socket ip = socket.gethostbyname('www.google.com')`  
`print ip`

### Server Socket Methods

Method	Description
s.bind()	This method binds address (hostname, port number pair) to socket.
s.listen()	This method sets up and start TCP listener.
s.accept()	This passively accept TCP client connection, waiting until connection arrives (blocking).

### Client Socket Methods

Method	Description
s.recv()	This method receives message
s.send()	This method transmits message
s.recvfrom()	This method receives message

#### Server :

A server has a bind() method which binds it to a specific ip and port so that it can listen to incoming requests on that ip and port. A server has a listen() method which puts the server into listen mode. This allows the server to listen to incoming connections. And last a server has an accept() and close() method. The accept method initiates a connection with the client and the close method closes the connection with the client.

#### Client :

Now we need something with which a server can interact. We could connect to the server like this just to know that our server is working. Type these commands in the terminal: # start the server \$ python server.py # keep the above terminal open # now open another terminal and type: \$ telnet localhost 12345

**Conclusion:** From this experiment we studied the client server communication on a network with the help of socket.

#### Code:

##### Server Side:

```
import socket
```

```
HOST = '127.0.0.1'
```

```
PORT = 65432
```

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

```
    s.bind((HOST, PORT))
```

```
s.listen()

conn, addr = s.accept()

with conn:

    print('Connected by', addr)

    while True:

        data = conn.recv(1024)

        if not data:

            break

        conn.sendall(data)
```

#### Client Side:

```
import socket

HOST = '127.0.0.1'

PORT = 65432

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:

    s.connect((HOST, PORT))

    s.sendall(b'Hello, world')

    data = s.recv(1024)

print('Received', repr(data))
```

#### **Output:**

##### Server Side:

```
Command Prompt
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\adnan>cd C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp12

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp12>python server.py
Connected by ('127.0.0.1', 58842)

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp12>
```

Client Side:

```
Command Prompt
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\adnan>cd C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp12

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp12>python client.py
Received b'Hello, world'

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp12>
```

## EXPERIMENT NO. 8

**Aim:** Programs on Threading using python.

**Resources Required:** Consumables – Printer Pages for printouts.

### Theory:

#### Single Thread:

#### Code:

```
import logging
import threading
import time

def thread_function(name):
    logging.info("Thread %s: starting", name)
    time.sleep(2)
    logging.info("Thread %s: finishing", name)

if __name__ == "__main__":
    format = "%(asctime)s: %(message)s"
    logging.basicConfig(format=format, level=logging.INFO,
                        datefmt="%H:%M:%S")

    logging.info("Main : before creating thread")
    x = threading.Thread(target=thread_function, args=(1,))
    logging.info("Main : before running thread")
    x.start()
    logging.info("Main : wait for the thread to finish")
    # x.join()
    logging.info("Main : all done")
```

#### Output:

```
$ ./single_thread.py
Main : before creating thread
Main : before running thread
Thread 1: starting
Main : wait for the thread to finish
Main : all done
Thread 1: finishing
```

#### Multithread:

#### Code:

```
import logging
import threading
```

```

import time

def thread_function(name):
    logging.info("Thread %s: starting", name)
    time.sleep(2)
    logging.info("Thread %s: finishing", name)

if __name__ == "__main__":
    format = "%(asctime)s: %(message)s"
    logging.basicConfig(format=format, level=logging.INFO,
                        datefmt="%H:%M:%S")

    threads = list()
    for index in range(3):
        logging.info("Main : create and start thread %d.", index)
        x = threading.Thread(target=thread_function, args=(index,))
        threads.append(x)
        x.start()

    for index, thread in enumerate(threads):
        logging.info("Main : before joining thread %d.", index)
        thread.join()
        logging.info("Main : thread %d done", index)

```

### Output:

```

$ ./multiple_threads.py
Main : create and start thread 0.
Thread 0: starting
Main : create and start thread 1.
Thread 1: starting
Main : create and start thread 2.
Thread 2: starting
Main : before joining thread 0.
Thread 2: finishing
Thread 1: finishing
Thread 0: finishing
Main : thread 0 done
Main : before joining thread 1.
Main : thread 1 done
Main : before joining thread 2.
Main : thread 2 done

```



## EXPERIMENT NO.9

AIM: Exploring basics of NumPy Methods.

RESOURCES REQUIRED: Consumables – Printer Pages for printouts.

### THEORY:

#### The array Method

To create a one-dimensional NumPy array, we can simply pass a Python list to the `array` method. Check out the following script for an example:

#### The arange Method

Another commonly used method for creating a NumPy array is

the `arange` method. This method takes the start index of the array, the end index, and the step size (which is optional).

#### The zeros Method

Apart from generating custom arrays with your pre-filled data, you can also create NumPy arrays with a simpler set of data. For instance, you can use

the `zeros` method to create an array of all zeros

#### The ones Method

Similarly, you can create one-dimensional and two-dimensional arrays of all ones using the `ones` method

#### The linspace Method

Another very useful method to create NumPy arrays is the `linspace` method. This method takes three arguments: a start index, end index, and the number of linearly-spaced numbers that you want between the specified range. For instance, if the first index is 1, the last index is 10 and you need 10 equally spaced elements within this range, you can use the `linspace` method

#### The eye Method

The `eye` method can be used to create an identity matrix, which can be very useful to perform a variety of operations in linear algebra. An identity matrix is a matrix with zeros across rows and columns except the diagonal. The diagonal values are all ones.

#### The random Method

Often times you will need to create arrays with random numbers. You can use the `rand` function of NumPy's random module to do so

#### Reshaping NumPy Array

Using NumPy you can convert a one-dimensional array into a two-dimensional array using the reshape method.

### Finding Max/Min Values

You can use min/max functions to easily find the value of the smallest and largest number in your array.

**Conclusion:** Hence we have successfully study the basic methods of numpy.

### Code:

```
import numpy as np

#array method x = [2, 3, 4, 5, 6]
nums = np.array([2, 3, 4, 5, 6])
print(type(nums))

#arrange method
nums=np.arange(2,10)
print(nums)

#zero method
nums=np.zeros((5,4))
print(nums)

#ones method
nums=np.ones((5,4))
print(nums)

#linspace methods
nums=np.linspace(1,10,5)
print(nums)

#eye method
nums=np.eye(4)
print(nums)

#random method
nums = np.random.rand(2,3)
```

```

print(nums)

#reshaping array

num=np.arange(1,17)

num2=num.reshape(4,4)

print(num2)

#findind max,min value

random=np.random.randint(1,100,5)

print(random)

xmin=random.min()

print(xmin)

xmax=random.max()

print(xmax)

```

### Output:

```

PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'> cd numpy
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\numpy> python numpyprac.py
<class 'numpy.ndarray'>
[2 3 4 5 6 7 8 9]
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
[ 1.  3.25  5.5  7.75 10. ]
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
[[0.54228272 0.87313139 0.37389622]
 [0.36440035 0.72808304 0.60440469]]
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
[57 63 16 30 40]
16
63
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\numpy> 

```

## EXPERIMENT NO.10

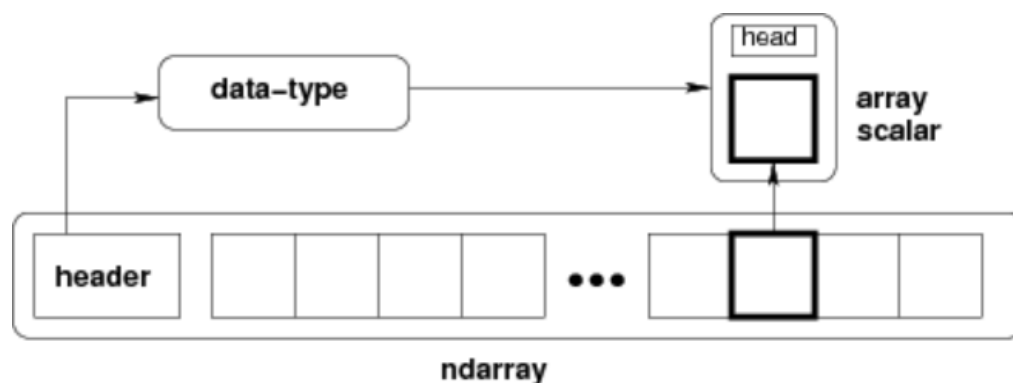
**AIM:** Program to demonstrate use of NumPy: Array objects.

**RESOURCES REQUIRED:** Consumable-printout pages

### THEORY:

NumPy provides an N-dimensional array type, the `ndarray`, which describes a collection of “items” of the same type. The items can be `indexed` using for example N integers.

All `ndarrays` are **homogeneous**: every item takes up the same size block of memory, and all blocks are interpreted in exactly the same way. How each item in the array is to be interpreted is specified by a separate data-type object, one of which is associated with every array. In addition to basic types (integers, floats, *etc.*), the data type objects can also represent data structures.



**Conclusion:** Hence, we have successfully the study concept of Numpy: Array Objects

### Code:

```
import numpy as np

x = int(input("enter number of rows: "))
y = int(input("enter number of columns: "))

matrix = lambda : [list(map(int,input().strip().split())) for _ in range(x)]
for i in range(2):
    print("matrix: ", i + 1)
    if i==0:
        mat1=np.array(matrix())
    elif i==1:
        mat2=np.array(matrix())

print("matrix 1 is: ")
print(mat1)
print("matrix 2 is: ")
```

```
print(mat2)
res = np.dot(mat1, mat2)

print("Multiplication of matrix 1 and 2 is : \n",res)
```

### Output:

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\numpy> python numpyobj.py
enter number of rows: 3
enter number of columns: 3
matrix: 1
56 87 69
45 12 78
420 56 59
matrix: 2
1 0 0
0 1 0
0 0 1
matrix 1 is:
[[ 56  87  69]
 [ 45  12  78]
 [420  56  59]]
matrix 2 is:
[[1 0 0]
 [0 1 0]
 [0 0 1]]
Multiplication of matrix 1 and 2 is :
[[ 56  87  69]
 [ 45  12  78]
 [420  56  59]]
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\numpy> □
```

## EXPERIMENT NO.11

**AIM:** Program to demonstrate Data Series and Data Frames using Pandas.

**RESOURCES REQUIRED:** Consumables – Printer Pages for printouts.

### THEORY:

#### Series Initialized with Dictionaries

With pandas we can also use the dictionary data type to initialize a Series. This way, we will not declare an index as a separate list but instead use the built-in keys as the index.

**DataFrames** are 2-dimensional labeled data structures that have columns that may be made up of different data types.

DataFrames are similar to spreadsheets or SQL tables. In general, when you are working with `pandas`, DataFrames will be the most common object you'll use.

To understand how the `pandas` DataFrame works, let's set up two Series and then pass those into a DataFrame.

With those two Series set up, let's add the DataFrame to the bottom of the file. In our example, both of these Series have the same index labels, but if you had Series with different labels then missing values would be labelled NaN.

This is constructed in such a way that we can include column labels, which we declare as keys to the Series' variables.

**Conclusion:** Hence we have successfully study and implemented the concept of Data series and Frames in pandas.

### Code and Output:

```
In [1]: import pandas as pd

In [2]: raw_data = {"name": ['Bulbasaur', 'Charmander', 'Squirtle', 'Caterpie'],
                    "evolution": ['Ivysaur', 'Charmeleon', 'Wartortle', 'Metapod'],
                    "type": ['grass', 'fire', 'water', 'bug'],
                    "hp": [45, 39, 44, 45],
                    "pokedex": ['yes', 'no', 'yes', 'no']}

In [3]: pokemon = pd.DataFrame(raw_data)
        pokemon.head()

Out[3]:
```

	name	evolution	type	hp	pokedex
0	Bulbasaur	Ivysaur	grass	45	yes
1	Charmander	Charmeleon	fire	39	no
2	Squirtle	Wartortle	water	44	yes
3	Caterpie	Metapod	bug	45	no

```


In [4]: pokemon = pokemon[['name', 'type', 'hp', 'evolution', 'pokedex']]
        pokemon

Out[4]:
```

	name	type	hp	evolution	pokedex
0	Bulbasaur	grass	45	Ivysaur	yes
1	Charmander	fire	39	Charmeleon	no
2	Squirtle	water	44	Wartortle	yes
3	Caterpie	bug	45	Metapod	no

```
In [5]: pokemon['place'] = ['park','street','lake','forest']
pokemon
```

```
Out[5]:
```

	name	type	hp	evolution	pokedex	place
0	Bulbasaur	grass	45	Ivysaur	yes	park
1	Charmander	fire	39	Charmeleon	no	street
2	Squirtle	water	44	Wartortle	yes	lake
3	Caterpie	bug	45	Metapod	no	forest

```
In [6]: pokemon.dtypes
```

```
Out[6]: name      object
         type      object
         hp        int64
         evolution object
         pokedex   object
         place     object
         dtype: object
```

```
In [ ]:
```

```
In [7]: ds = pd.Series(pokemon.items())
ds
```

```
Out[7]: 0      (name, [Bulbasaur, Charmander, Squirtle, Cater...
         1              (type, [grass, fire, water, bug])
         2              (hp, [45, 39, 44, 45])
         3      (evolution, [Ivysaur, Charmeleon, Wartortle, M...
         4              (pokedex, [yes, no, yes, no])
         5      (place, [park, street, lake, forest])
         dtype: object
```

```
In [9]: ds[0]
```

```
Out[9]: ('name',
0      Bulbasaur
1      Charmander
2      Squirtle
3      Caterpie
Name: name, dtype: object)
```

```
In [ ]:
```

## EXPERIMENT NO.12

**AIM:** Program to send email and read content of URL.

**RESOURCES REQUIRED:** Email-id and password with Less Security Access –ON; URL link

### THEORY:

#### Send Mail:

Simple Mail Transfer Protocol (SMTP) is a protocol, which handles sending e-mail and routing e-mail between mail servers.

Python provides smtplib module, which defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.

Here is a simple syntax to create one SMTP object, which can later be used to send an e-mail –

```
import smtplib
```

```
smtpObj = smtplib.SMTP( [host [, port [, local_hostname]]] )
```

Here is the detail of the parameters –

host – This is the host running your SMTP server. You can specify IP address of the host or a domain name like tutorialspoint.com. This is optional argument.

port – If you are providing host argument, then you need to specify a port, where SMTP server is listening. Usually this port would be 25.

local\_hostname – If your SMTP server is running on your local machine, then you can specify just localhost as of this option.

An SMTP object has an instance method called sendmail, which is typically used to do the work of mailing a message. It takes three parameters –

The sender – A string with the address of the sender.

The receivers – A list of strings, one for each recipient.

The message – A message as a string formatted as specified in the various RFCs.

---

#### Read content of URL:

The Internet is an enormous source of data and, often, websites will offer a RESTful API endpoints (URLs, URIs) to share data via HTTP requests. HTTP requests are composed of methods like GET, POST, PUT, DELETE, etc. to manipulate and access resources or data. Often, websites require a registration process to access RESTful APIs or offer no API at all. So, to simplify the process, we can also download the data as raw text and format it. For instance, downloading content from a personal blog or profile information of a GitHub user without any registration. This guide will explain the process of making web requests in python using Requests package and its various features.

**Conclusion:** Hence we have successfully send mail using SMTP Library.



## SMTP:

### **Code:**

```
import smtplib, ssl

port = 465

smtp_server = "smtp.gmail.com"

sender_email = "ww2232785@gmail.com"

receiver_email = "aa2232786@gmail.com"

password = "You expected password but it was me DIO!!!!!!"

message = """\n
Subject: Hi there

This message is sent from bot.""

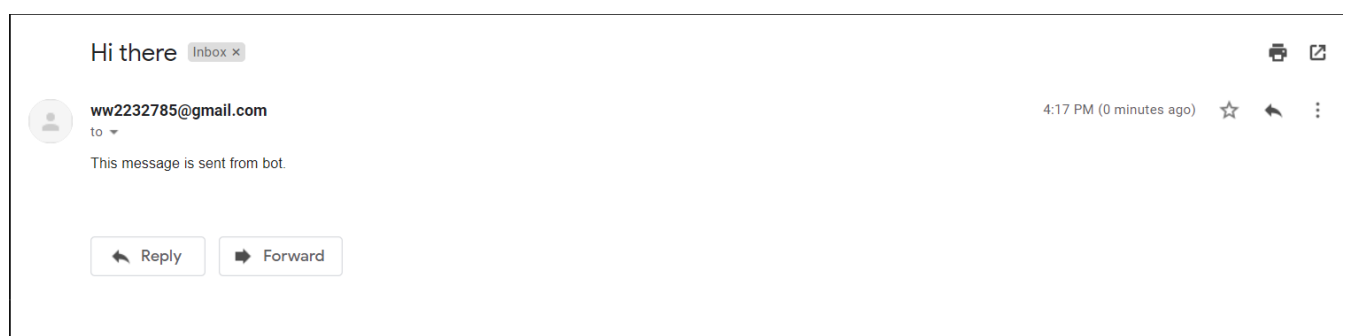
context = ssl.create_default_context()

with smtplib.SMTP_SSL(smtp_server, port, context=context) as server:

    server.login(sender_email, password)

    server.sendmail(sender_email, receiver_email, message)
```

### **Output:**



## URL:

### **Code:**

```
import requests

import json

import pandas as pd

url = "https://covid-19-data.p.rapidapi.com/country"

headers = {
```

```

    'x-rapidapi-key': "Generated API Key from rapidapi here",
    'x-rapidapi-host': "covid-19-data.p.rapidapi.com"
}

response = requests.request("GET",url,headers=headers,params={"name":"india"}).json()

ds = pd.Series(response)

print(*[x for x in ds.items()])

```

### Output:

```

PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python> cd sm
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python\sm> python exp12.py
(0, {'country': 'India', 'code': 'IN', 'confirmed': 27369093, 'recovered': 24633951, 'critical': 8944, 'deaths': 315263, 'latitude': 20.593684, 'longitude': 78.96288,
'lastChange': '2021-05-27T05:47:10+02:00', 'lastUpdate': '2021-05-27T12:30:03+02:00'})
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python\sm> 

```

## ASSIGNMENT NO. 1

### Exception handling

A. Write a program to demonstrate exception handling using try, multiple exception and finally.

**Code:**

```
import math

import warnings

warnings.filterwarnings("ignore", category=DeprecationWarning)

number_list = [6,-9,420.69,'banana']

for number in number_list:

    try:

        number_factorial = math.factorial(number)

    except TypeError:

        print("Factorial is not supported for given input type. \n")

    except ValueError:

        print(f"Factorial only accepts whole values. {number} is not a whole value. \n")

    else:

        print(f"The factorial of {number} is {number_factorial}\n")

    finally:

        print("Release any resources in use. \n")
```

**Output:**

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp9> python exceptionhandling.py
The factorial of 6 is 720

Release any resources in use.

Factorial only accepts whole values. -9 is not a whole value.

Release any resources in use.

Factorial only accepts whole values. 420.69 is not a whole value.

Release any resources in use.

Factorial is not supported for given input type.

Release any resources in use.
```

B. Write a python program to create user defined exception.

**Code:**

```
#numberGuessing

class ValueTooSmallError(Exception):

    #Raised when the input value is too small

    pass

class ValueTooLargeError(Exception):

    #Raised when the input value is too large

    pass

number = 10

while True:

    try:

        i_num = int(input("Guess the number: "))

        if i_num < number:

            raise ValueTooSmallError

        elif i_num > number:
```

```

        raise ValueErrorTooLargeError

    break

except ValueErrorTooSmallError:

    print("This value is too small, try again!")

    print()

except ValueErrorTooLargeError:

    print("This value is too large, try again!")

    print()

print("Congratulations! You guessed it correctly.")

```

#### Output:

```

PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp9> python userdefinedexception.py
Guess the number: 56
This value is too large, try again!

Guess the number: 99
This value is too large, try again!

Guess the number: 69
This value is too large, try again!

Guess the number: 7
This value is too small, try again!

Guess the number: 9
This value is too small, try again!

Guess the number: 11
This value is too large, try again!

Guess the number: 10
Congratulations! You guessed it correctly.

```

## ASSIGNMENT NO. 2

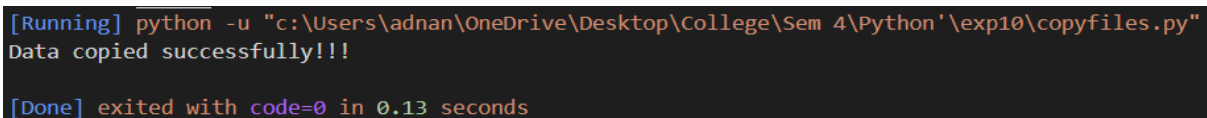
### Exploring files and directories.

A. Python program to read the content of file and write in another file.

**Code:**

```
file1 = open("dio.txt", "w")
file1.write("KONO DIO DAAA")
file1.close()
file2 = open("wurdo.txt", "w")
file2.write("You expected Output but it was me DIO")
file2.close()
file1 = open("dio.txt", "r")
file2 = open("wurdo.txt", "r")
file3 = open("output.txt", "w")
file3.write(file1.read())
file3.write(file2.read())
file1.close()
file2.close()
file2.close()
print("Data copied successfully!!!")
```

**Output:**



```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp10\copyfiles.py"
Data copied successfully!!!
[Done] exited with code=0 in 0.13 seconds
```

**Content of files:**

DIO.txt: KONO DIO DAAA

wurdo.txt: You expected Output but it was me DIO

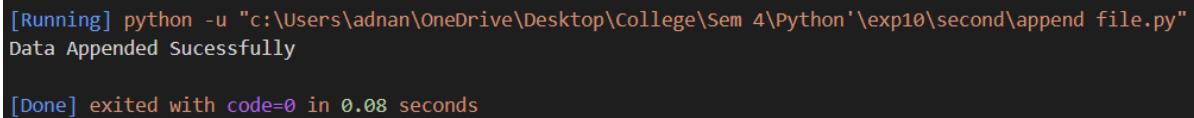
output.txt: KONO DIO DAAAYou expected Output but it was me DIO

B. Program to append data to existing file and then display entire file.

**Code:**

```
f=open("Adnan.txt","a+")  
f.write(" Giorno Giovanna")  
print("Data Appended Sucessfully")  
f.close()
```

**Output:**



```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp10\second\append file.py"  
Data Appended Sucessfully  
[Done] exited with code=0 in 0.08 seconds
```

**Content of a file:**

Before appending: Adnan Shaikh

After appending: Adnan Shaikh Giorno Giovanna

C. Program to count number of lines, words and characters in a file.

**Code:**

```
file = open("Count.txt", "r")

number_of_lines = 0

number_of_words = 0

number_of_characters = 0

for line in file:

    line = line.strip("\n")

    words = line.split()

    number_of_lines += 1

    number_of_words += len(words)

    number_of_characters += len(line)

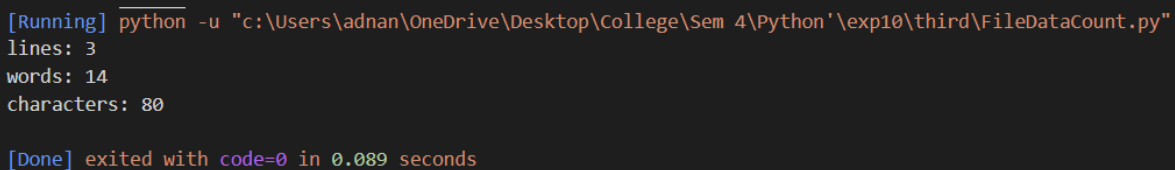
file.close()

print("lines:", number_of_lines)

print("words:", number_of_words)

print("characters:", number_of_characters)
```

Output:



```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp10\third\FileDataCount.py"
lines: 3
words: 14
characters: 80

[Done] exited with code=0 in 0.089 seconds
```

**Content of a file:**

Hello, this is Adnan

Welcome to my woooooorld

Ok, understandable have a great day.



D. Program to display file available in current directory.

**Code:**

```
import os

from pathlib import Path

if os.path.exists("Adnan.txt"):

    print("YES, The file does exist")

else:

    print("NO, The file does not exist\nCreating a new file called Adnan.txt")

    f1 = open("Adnan.txt", "w+")

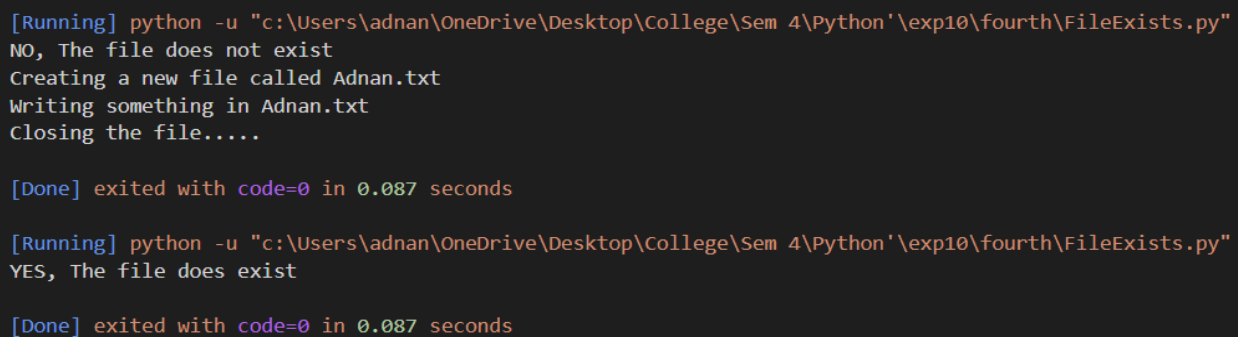
    print("Writing something in Adnan.txt")

    f1.write("Well well what do we have we here")

    print("Closing the file.....")

    f1.close()
```

**Output:**



```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp10\fourth\FileExists.py"
NO, The file does not exist
Creating a new file called Adnan.txt
Writing something in Adnan.txt
Closing the file.....

[Done] exited with code=0 in 0.087 seconds

[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp10\fourth\FileExists.py"
YES, The file does exist

[Done] exited with code=0 in 0.087 seconds
```

**Content of a file:** Well well what do we have we here