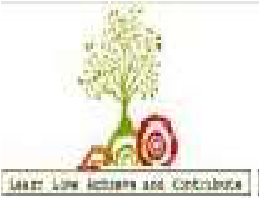
# Saraswati College Of Engineering, Kharghar Computer Department

**Saraswati College of Engineering, Kharghar**

**Name: Shaikh Adnan Shaukat Ali**

**Roll no: 55**

**Subject: Python Programming**

**Class/Sem: SE/IV**

Saraswati College Of Engineering, Kharghar

Department of Computer Engineering

S.E.

Skill Base Lab Course: Python Programming

-:Lab Manual:-

SCOE’s

# Saraswati College Of Engineering, Kharghar Computer Department

Saraswati College of Engineering, Kharghar

Vision:

To become center of excellence in Engineering education and research.

Mission:

To educate students to become quality technocrats for taking up challenges in all facets of life.

## Department of Computer Engineering

Vision:

To imprint knowledge to our students to excel in engineering culture and research and nurture them to become ethically rich professionals.

Mission:

To provide simulating learning environment with a technological orientation to maximize individual potential.

## SARASWATI EDUCATION SOCIETY’S SARASWATI COLLEGE OF ENGINEERING,

NAVI MUMBAI, KHARGHAR. COMPUTER ENGINEERING DEPARTMENT

Programme Outcomes (PO)

At the end of the program, a student will be able to:

1. Apply knowledge of mathematics, science and engineering.
2. Utilize the computer engineering knowledge in all domains, viz., health care, banking and Finance, other professions such as medical, law, etc.
3. Design and conduct experiments as well as to analyze and interpret data.
4. Analyze the problem, subdivide it into smaller tasks with well-defined interface for interaction among components, and complete the task within the specified time frame and financial constraints,
5. Design a system, component or process to meet the desired needs within realistic constraints such as economic, environmental, social, political and Ethical ability,
6. Design, implement, and evaluate secure hardware and/or software systems with assured quality and efficiency,
7. Communicate effectively the engineering solution to customers/users or peers,
8. Understand professional and ethical responsibilities,
9. Understand contemporary issues and to get engaged in lifelong learning by independently and continually expanding knowledge and abilities,
10. Function in multidisciplinary teams,
11. Identify, formulate and solve engineering problems.

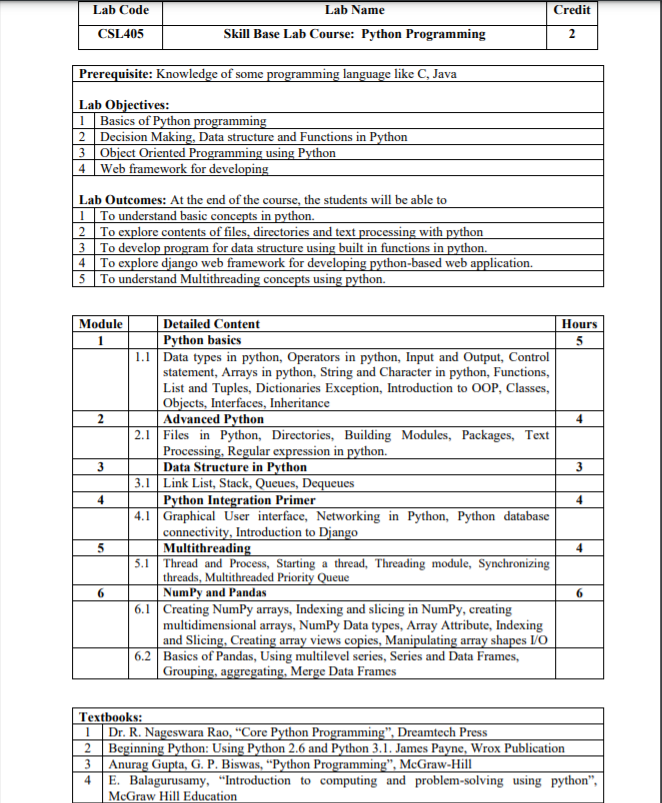
## SARASWATI EDUCATION SOCIETY’S

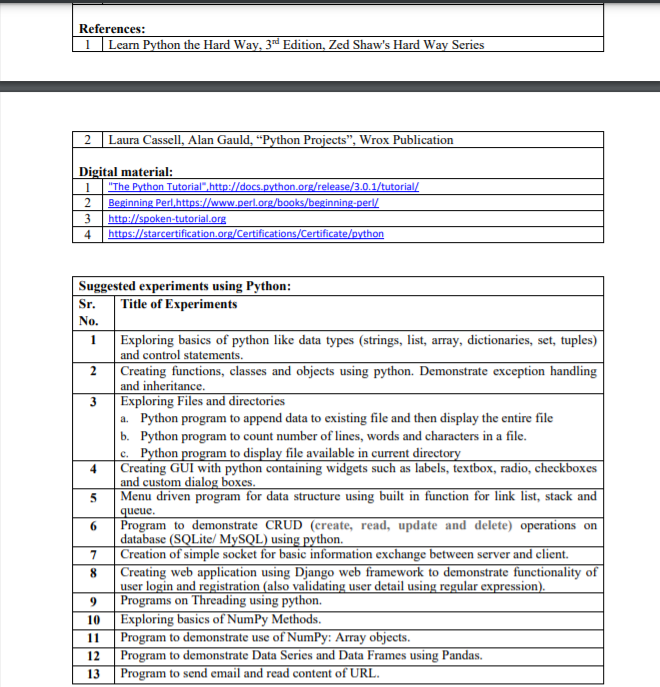
SARASWATI COLLEGE OF ENGINEERING, NAVI MUMBAI, KHARGHAR.

COMPUTER ENGINEERING DEPARTMENT

Program Educational Objectives (PEO)

1. To prepare the candidate for a successful career in the industry and make him acquainted with the latest software and hardware,
2. To enable student to work productively as computer engineers, including supportive teamwork and leadership roles on multidisciplinary teams,
3. Graduates are prepared to be responsible computing professionals in their own area of interest,
4. To provide the candidate with a sound foundation in mathematics, software technologies, database technologies, networking, hardware and to prepare them for post graduate studies and research programs,
5. To promote the awareness of lifelong learning among students and to introduce them to professional ethics and codes of professional practice,
6. To demonstrate effective communication skills in oral, written and electronic media.





## SARASWATI COLLEGE OF ENGINEERING COMPUTER ENGINEERING DEPARTMENT

Subject: Python Programming Class/Sem: SE/IV

Name of the Laboratory: Skill Base Lab Course: Python Programming Year: 2020-21

LIST OF EXPERIMENTS

|  |  |
| --- | --- |
| Experiment  No. | Name of the Experiment |
| 1 | Program to swap two numbers and check if first number is positive or negative or zero. |
| 2 | Program to check if number and string is palindrome and find factorial of input number. |
| 3 | Menu driven program to demonstrate use of list in python.   1. Put even and odd elements into two different lists. 2. Merge and sort two lists. 3. Update first element with x value and delete middle element of list. 4. Find max and min element from list. 5. Add n names into the existing number list and check if word python is present in list. |
| 4 | Menu driven program to demonstrate use of tuples in python.   1. Add and show N student roll number, name and 3 subject marks in a list of tuples. 2. Display student roll number and marks whose name is python. 3. Demonstrate nested tuple and sort nested tuple by name. |
| 5 | Menu driven program to demonstrate use of set in python   1. Accept two strings from user 2. Display common letters in two input strings(set intersection) 3. Display letters which are in the first string but not in the second.(set difference) 4. Display set of all letters of both the strings.(set union) 5. Displays letters which are in the two strings but not in both.(symmetric difference) |
| 6 | Menu driven program to demonstrate use of dictionary in python   1. Create key/value pair dictionary. 2. Update/concatenate and delete item of existing dictionary. 3. Find a key and print its value. 4. Map two list into dictionary |
| 7 | Design an employee class using python for reading and displaying the employee information. |

|  |  |
| --- | --- |
| 8 | Program to demonstrate single and multiple inheritance in python (with method overloading and overriding) |
| 9 | Exception handling   1. Write a program to demonstrate exception handling using try, multiple exception and finally. 2. Write a python program to create user defined exception. |
| 10 | Exploring files and directories.   1. Python program to read the content of file and write in another file. 2. Program to append data to existing file and then display entire file. 3. Program to count number of lines, words and characters in a file. 4. Program to display file available in current directory. |
| 11 | Create a package and module for data structure :stack and queues. |
| 12 | Creation of simple socket for basic info exchange between server and client. |
| 13 | Connecting to database and executing basic commands. |

|  |  |
| --- | --- |
| H/W Requirement | P-IV and above, Ram 128 MB, Printer, Internet Connection |
| S/W Requirement | Python IDLE |

Practical Incharge H. O. D.

|  |  |  |
| --- | --- | --- |
| EXPERIMENT NO. 1 | | |
|  | | |
| Aim | : | Program to swap two numbers and check if first number is positive or negative or zero |
| Resources  Required | : | Consumables – Printer Pages for printouts. |
| Theory | : | Python is a general-purpose interpreted, interactive, object-oriented, and high- level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).  There are 3 ways of executing a Python program   1. Using Python’s command line window. 2. Using Python’s IDLE graphics window. 3. Directly from System prompt.   The first two are called interactive modes where we can type the program one line at a time and the complier executes it immediately. The last one is called non-interactive mode where we ask complier to execute our program after typing the entire program. Programmers can store Python script source code in a file with the ’.py’ extension, and use the interpreter to execute the contents of the file. To execute the script by the interpreter, you have to tell the interpreter the name of the file. Decision making is required when we want to execute a code only if a certain condition is satisfied. The if…elif…else statement is used in Python for decision making. Syntax for if statement as follows:   * 1. if expression:   //execute your code   * 1. if expression:   //execute your code  else:  //execute your code   * 1. if expression:   //execute your code elif expression:  //execute your code  else:  //execute your code |
| Conclusion | : | We have studied how to run python program and swapping of two |

|  |  |  |
| --- | --- | --- |
|  |  | numbers, control statement if else for find the given number is positive, negative or zero.  By Student - Adnan Shaikh |

**Code:**

num1, num2 = map(int,input("Enter Your Numbers: ").strip().split())

print(f'Bepfore Swapping: {num1}, {num2}')

num1, num2 = num2, num1

print(f'After Swapping: {num1}, {num2}')

num1, num2 = num2, num1

if num1 > 0:

print("First number is positive!!!")

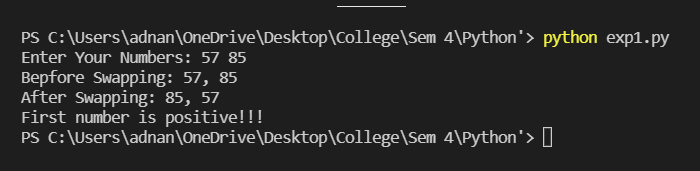
elif num1 < 0:

print("First number is negative!!!")

else:

print("First number is zero!!!")

**Output:**



|  |  |  |
| --- | --- | --- |
| EXPERIMENT NO. 2 | | |
|  | | |
| Aim | : | : Menu driven program to check if number and string is palindrome  and find factorial of input number |
| Resources  Required | : | Consumables – Printer Pages for printouts. |
| Theory | : | Functions in python.  The four steps to defining a function in Python are the following:   1. Use the keyword def to declare the function and follow this up with the function name. 2. Add arguments to the function: they should be within the parentheses of the function. End your line with a colon. 3. Add statements that the functions should execute. 4. End your function with a return statement if the function should output something. Without the return statement, your function will return an object None. 5. docstring is short for documentation string. It is used to explain in brief, what a function does.   Syntax for functions in python:  def function\_name(parameters):  """docstring""" statement(s)  A palindrome is a string which is same read forward or backwards. For example: "dad" is the same in forward or reverse direction. Another example is "aibohphobia" which literally means, an irritable fear of palindromes. The factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is 1\*2\*3\*4\*5\*6 = 720. Factorial is not defined for negative numbers and the factorial of zero is one, 0! = 1. |
| Conclusion | : | We have studied how to write functions in python program to find palindrome and factorial of number.  By Student – Adnan Shaikh |

**Code**:

def factorial(n):

if n <= 1:

return 1

else:

return n\*factorial(n-1)

def palindrome(str):

if str == str[::-1]:

print(f"{str} is palindrome!!!")

else:

print(f"{str} is not a palindrome!!!")

if \_\_name\_\_ == "\_\_main\_\_":

while(1):

num = int(input("\*\*\*\*\*\*\*\*\*Menu Driven\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n1. To calculate factorial\n2. To check given number or string is palindrome!!!!\n3. Exit\n"))

if num == 1:

n = int(input("Enter your number: "))

z = factorial(n)

print(f"{n}! = {z}")

elif num == 2:

palindrome(input("Enter number or a string: "))

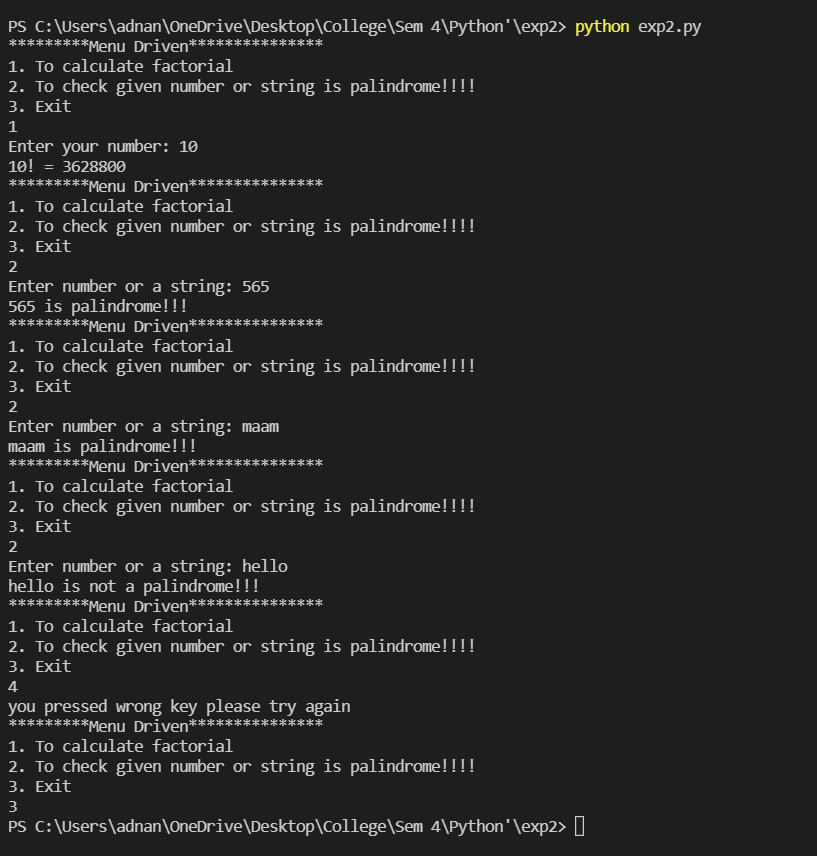
elif num == 3:

break

else:

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

**Output**:



|  |  |  |
| --- | --- | --- |
| EXPERIMENT NO. 3 | | |
|  | | |
| Aim | : | Write a Menu driven program to demonstrate use of list in python.   1. Put even and odd elements into two different lists. 2. Merge and sort two lists. 3. Update first element with x value and delete middle element of list. 4. Find max and min element from list. 5. Add n names into the existing number list and check if word python is present in list. |
| 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. |

|  |  |  |
| --- | --- | --- |
| Sr.  No. | Function | Description |
| 1 | cmp(list1,  list2) | Compares elements of both lists. |
| 2 | len(list) | Gives the total length of the list. |
| 3 | max(list) | Returns item from the list with max  value. |
| 4 | min(list) | Returns item from the list with min  value. |
| 5 | list(seq) | Converts a tuple into list. |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Python includes the following list functions –  Python includes following list methods: | | | | |
|  | Sr.No | Methods | Description |  |
| 1 | list.append(obj) | Appends object obj to list 2 |  |
| 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 | list.remove(obj) | Removes object obj from list |  |
| 8 | list.reverse() | Reverses objects of list in place |  |
| Conclusion | : | : Hence from above experiment student can understand that basic concept of list and list related various operation that create the list, update the list, merge the list etc.  By Student - Adnan Shaikh | | | | |

**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([])

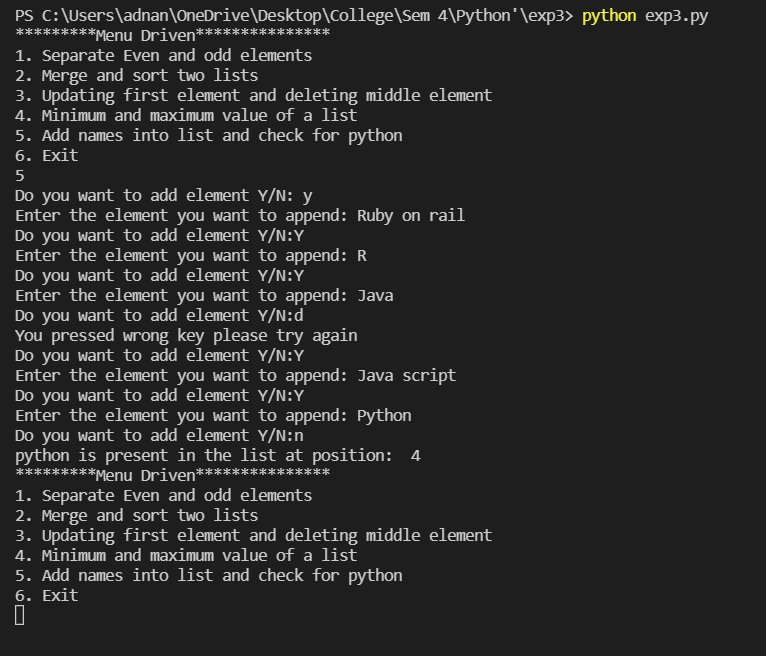
elif num == 6:

break

else:

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

**Output**:



|  |  |  |
| --- | --- | --- |
| EXPERIMENT NO. 4 | | |
|  | | |
| Aim | : | Menu driven program to demonstrate use of tuples in python.   1. Add and show N student roll number, name and 3 subject marks in a list of tuples. 2. Display student roll number and marks whose name is python. 3. Demonstrate nested tuple and sort nested tuple by name. |
| Resources  Required | : | Consumables – Printer Pages for printouts. |
| Theory | : | 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.).   1. Accessing Elements in a Tuple There are various ways in |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | which we can access the elements of a tuple.   1. 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 that (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.   1. 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.   1. 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.   1. 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 |
| Conclusion | : | Hence from this experiment we learned the various functions of tuples like delete, modify or insert elements of tuples since tuple are immutable, so for that how to create a new tuple and store the updated elements in it.  By Student - Adnan Shaikh | | | |

**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()

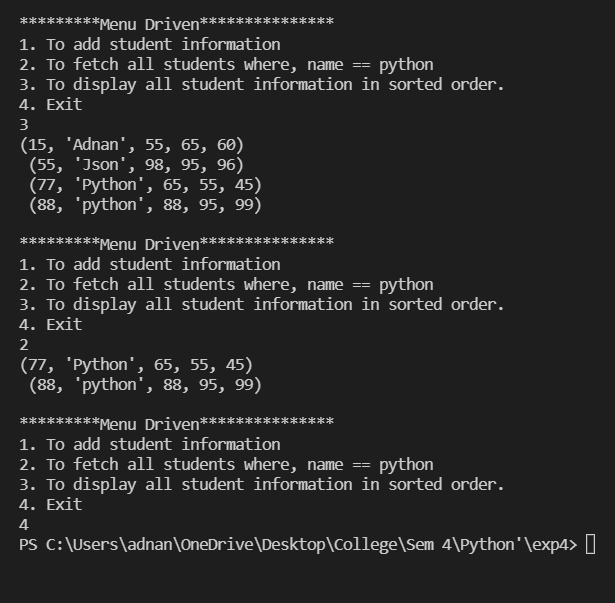
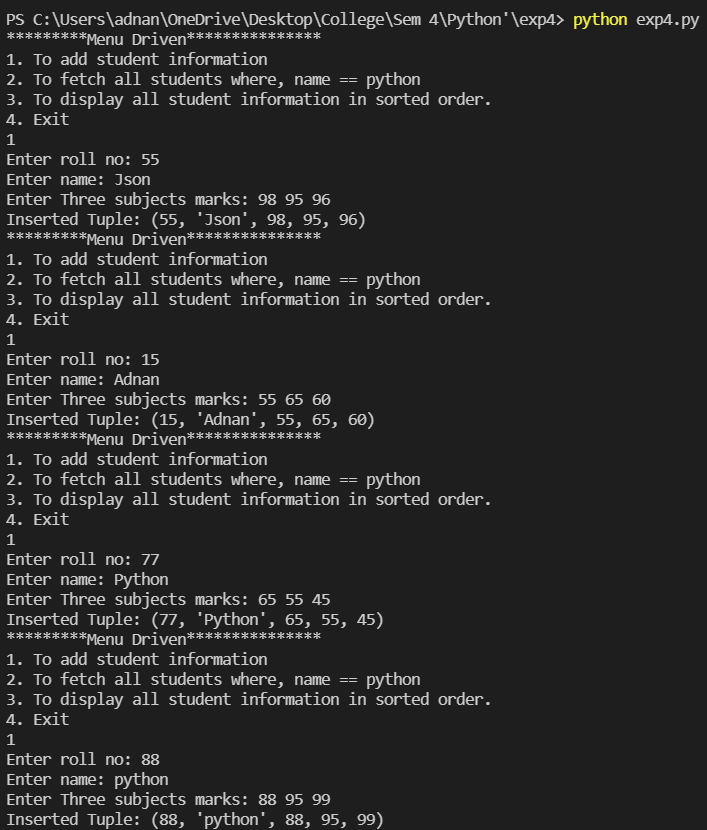
elif num ==4:

break

else:

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

**Output**:



|  |  |  |
| --- | --- | --- |
| EXPERIMENT NO. 5 | | |
|  | | |
| Aim | : | Write a menu driven program to demonstrate use of set in python   1. Accept two strings from user 2. Display common letters in two input strings (set intersection) 3. Display letters which are in the first string but not in the second (set difference) 4. Display set of all letters of both the strings (set union) 5. Displays letters which are in the two strings but not in both (symmetric difference) |
| Resources  Required | : | Consumables – Printer Pages for printouts. |
| Theory | : | 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. 5. clear() Method: Empties the whole set. |
| Conclusion | : | Hence from this experiment we learned the various operations of sets in python.  By Student - Adnan Shaikh |

**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()

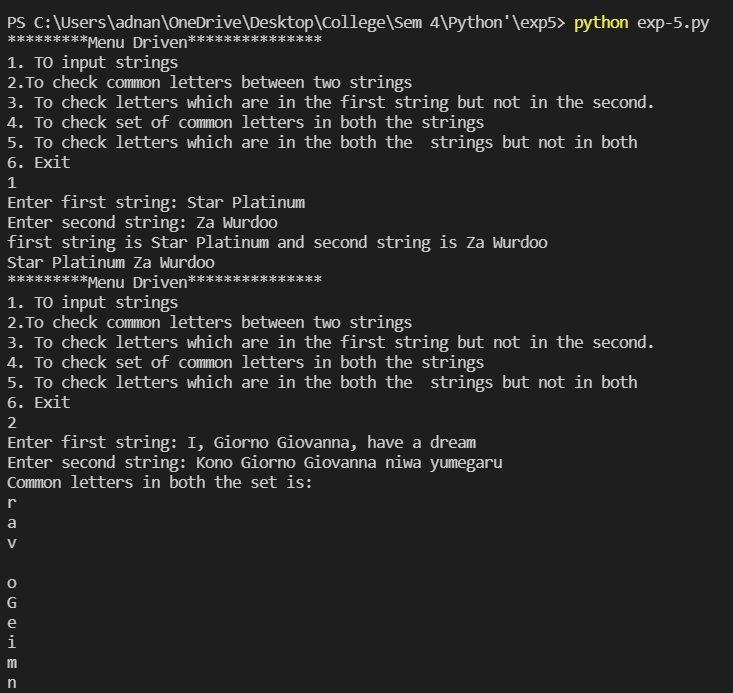
elif num==6:

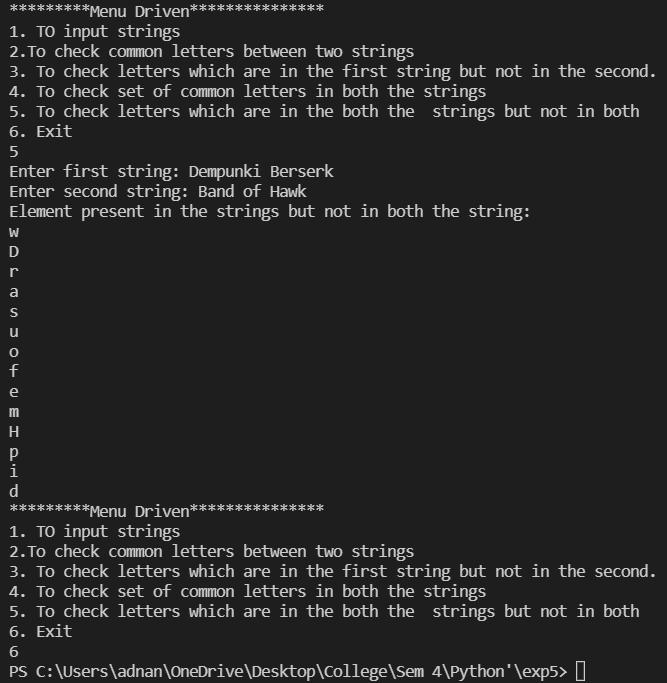
break

else:

print("Wrong key")

**Output**:





|  |  |  |
| --- | --- | --- |
| EXPERIMENT NO. 6 | | |
|  | | |
| Aim | : | Menu driven program to demonstrate use of dictionary in python   1. Create key/value pair dictionary. 2. Update/concatenate and delete item of existing dictionary. 3. Find a key and print its value. 4. Map two lists into dictionary. |
| Resources  Required | : | Consumables – Printer Pages for printouts. |
| Theory | : | 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.   1. 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.   1. 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.   1. 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 from this experiment we learned the various operation of dictionary like create, modify or insert elements of dictionary and how two lists can merge in the dictionary.  By Student - Adnan Shaikh |

**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 dicitonary

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\_linst():

key\_lst = []

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

# iterating till the range

for i in range(0, n):

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

key\_lst.append(element1)

value\_lst = []

# 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 dicitionary \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()

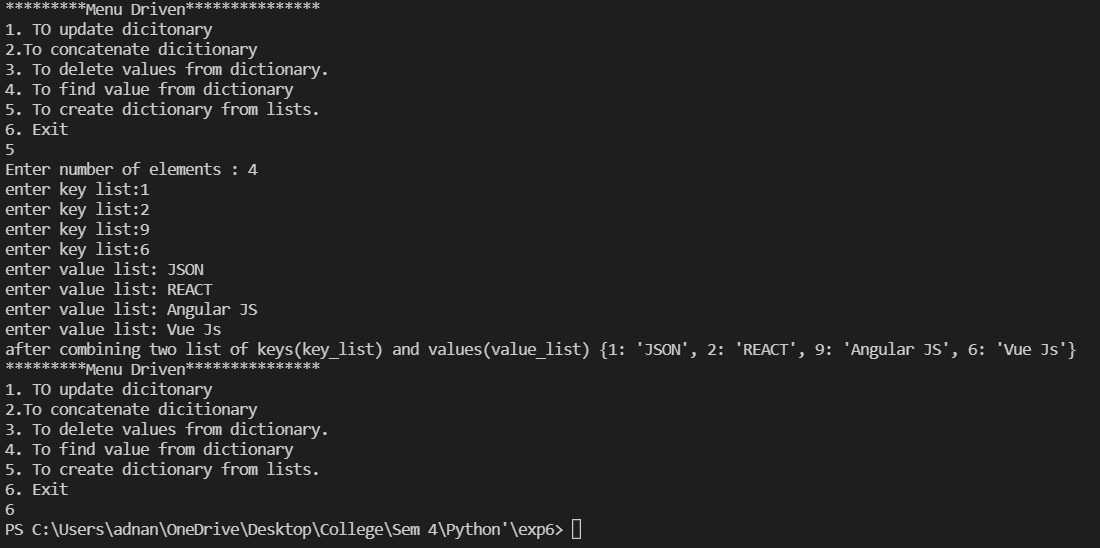
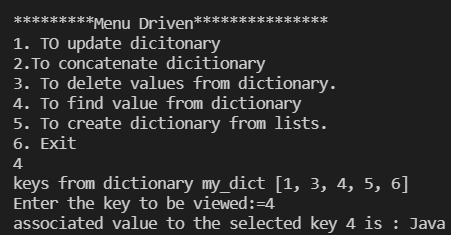
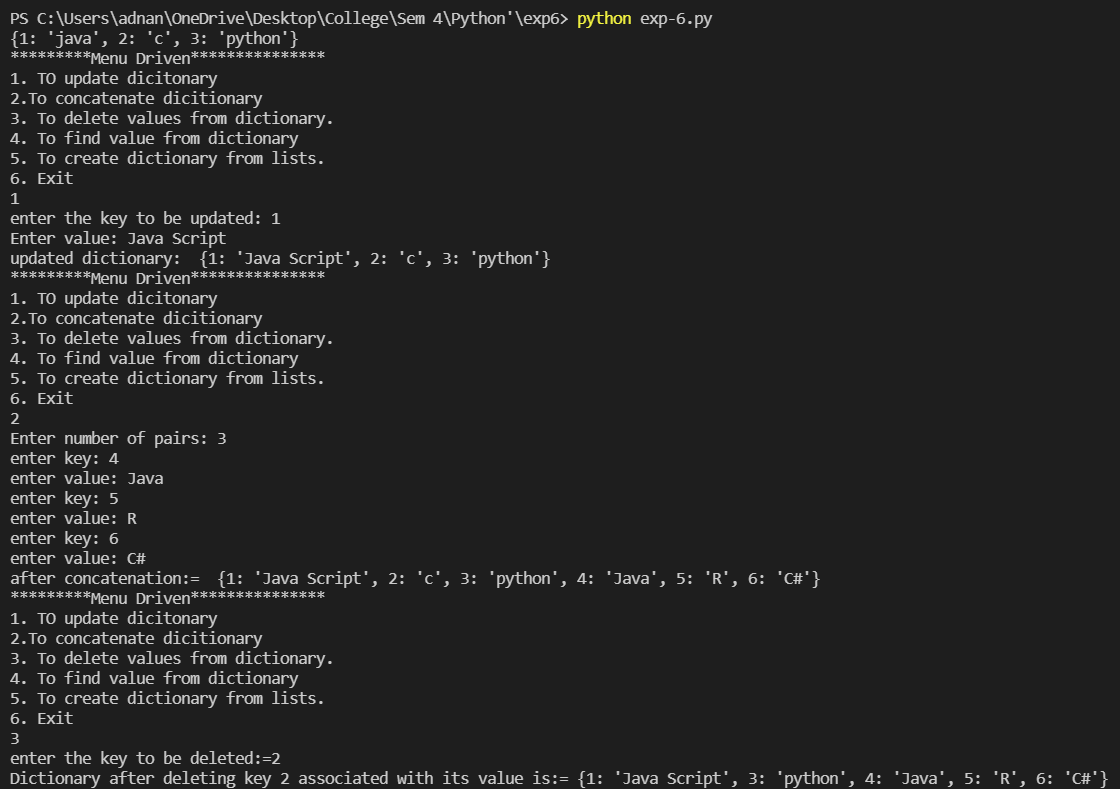
elif num==6:

break

else:

print("Wrong key")

**Output**:



|  |  |  |
| --- | --- | --- |
| EXPERIMENT NO. 7 | | |
|  | | |
| Aim | : | Design an employee class using python for reading and displaying  the employee information. |
| Resources  Required | : | Consumables – Printer Pages for printouts. |
| Theory | : | 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. |
| Conclusion | : | Hence from this experiment we learned how to create classes and its objects in python.  By Student - Adnan Shaikh |

**Code**:

class Employee:

count = 0

def \_\_init\_\_(self, name, desig, salary):

self.name = name

self.desig = desig

self.salary = salary

Employee.count += 1

def displayCount(self):

print("There are %d employees" % Employee.count)

def displayDetails(self):

print("Name:", self.name, ", Designation:", self.desig, ", Salary:", self.salary)

if \_\_name\_\_ == "\_\_main\_\_":

e1 = Employee("John", "Manager", 80000)

e2 = Employee("Mike", "Team Leader", 50000)

e3 = Employee("Ramesh", "Programmer", 30000)

e4 = Employee("Raj", "Assistant", 25000)

e4.displayCount()

print("Details of all employee:")

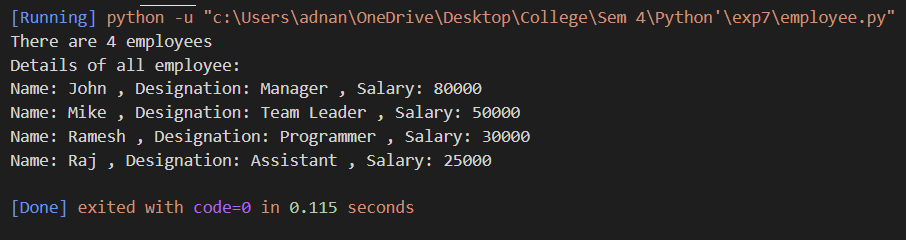
e1.displayDetails()

e2.displayDetails()

e3.displayDetails()

e4.displayDetails()

**Output**:



|  |  |  |
| --- | --- | --- |
| EXPERIMENT NO. 8 | | |
| Aim | : | Program to demonstrate single and multiple inheritance in python  (with method overloading and overriding) |
| Resources  Required | : | Consumables – Printer Pages for printouts. |
| Theory | : | 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.      1. Multiple Inheritances: A derived class with several base classes is called Multiple   Inheritance.     1. Hierarchical Inheritance: More than one class may inherit the features of one class this process is called as Hierarchical Inheritance.      1. 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 | : | From this experiment we studied how to write a class and use the concept of inheritance to achieve the specified requirements.  By Student - Adnan Shaikh |

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 = "Moooo"

def naam(self):

print("Name: ",self.name)

def voice(self):

print(f"{self.name} {self.voc}")

if \_\_name\_\_ == "\_\_main\_\_":

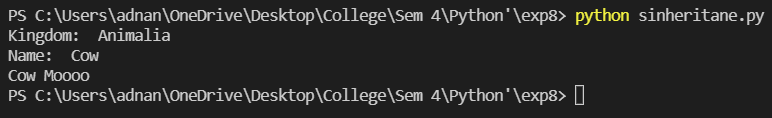
cow1 = Cow()

cow1.kingdom()

cow1.naam()

cow1.voice()

**Output**:



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\_\_":

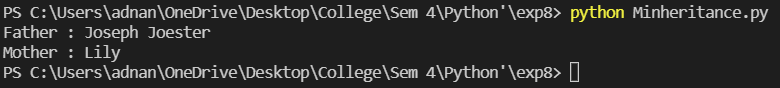
s1 = Son() # Object of Son class

s1.fathername = "Joseph Joester"

s1.mothername = "Lily"

s1.show\_parent()

**Output:**



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()

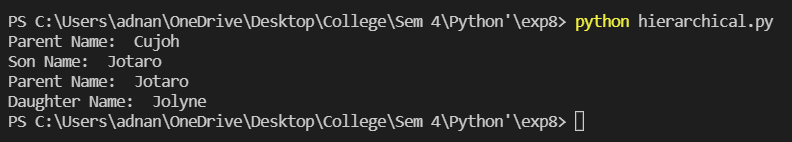
d1.childname = "Jolyne"

d1.parentname = "Jotaro"

d1.show\_parent()

d1.show\_child()

**Output**:



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"

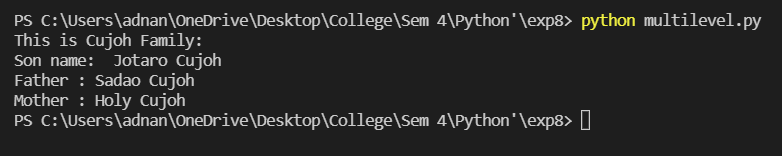
s1.fathername = "Sadao Cujoh"

s1.mothername = "Holy Cujoh"

s1.show\_family()

s1.show\_parent()

**Output**:



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EXPERIMENT NO. 9 | | | | | | |
|  | | | | | | |
| Aim | : | Exception handling   1. Write a program to demonstrate exception handling using try, multiple exception and finally. 2. Write a python program to create user defined exception. | | | | |
| Resources  Required | : | Consumables – Printer Pages for printouts. | | | | |
| Theory | : | Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them:   1. Exception Handling 2. 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. |  |

|  |  |  |
| --- | --- | --- |
| Conclusion | : | From this experiment we studied how to use standard exceptions and create user defined exceptions in python.  By Student - Adnan Shaikh |

**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")

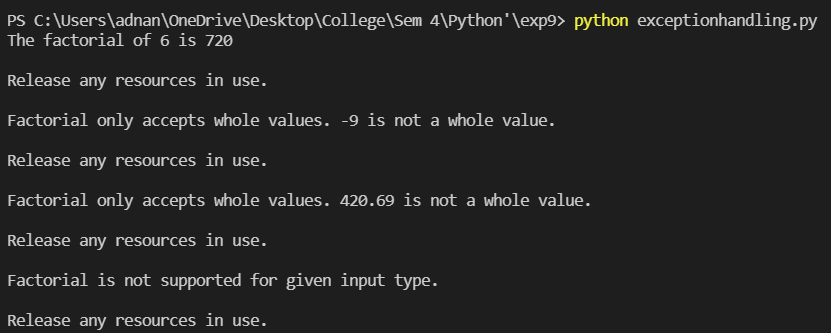
else:

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

finally:

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

**Output**:



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()

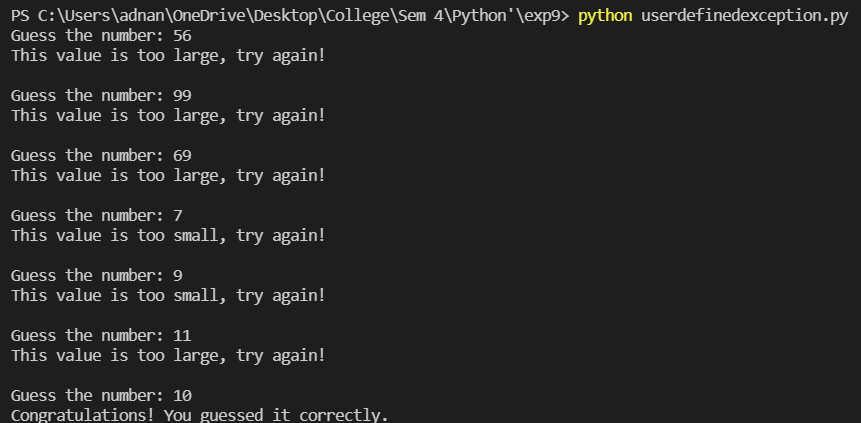
except ValueTooLargeError:

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

print()

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

**Output**:



|  |  |  |
| --- | --- | --- |
| EXPERIMENT NO. 10 | | |
|  | | |
| Aim | : | Exploring files and directories.   1. Python program to read the content of file and write in another file. 2. Program to append data to existing file and then display entire file. 3. Program to count number of lines, words and characters in a file. 4. 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:   1. to store huge amount of data When data is stored in a file, it is stored permanently. 2. It is possible to update the file data. 3. 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.text”, “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 from above experiment student will understand various file operations like creation of file, storing data in the files and retrieve data from the files in python.  By Student - Adnan Shaikh |

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())

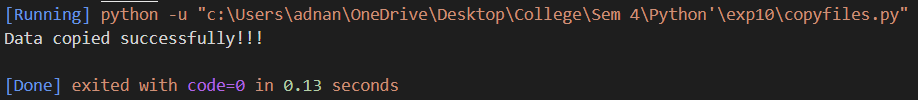
file1.close()

file2.close()

file2.close()

print("Data copied successfully!!!")

**Output**:



**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:**

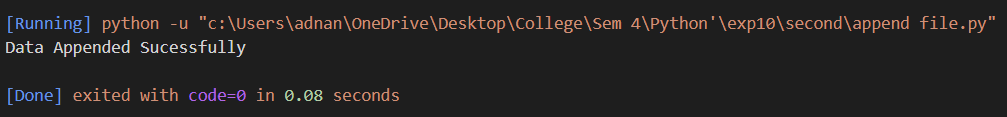
f=open("Adnan.txt","a+")

f.write(" Giorno Giovanna")

print("Data Appended Sucessfully")

f.close()

**Output:**

****

**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)

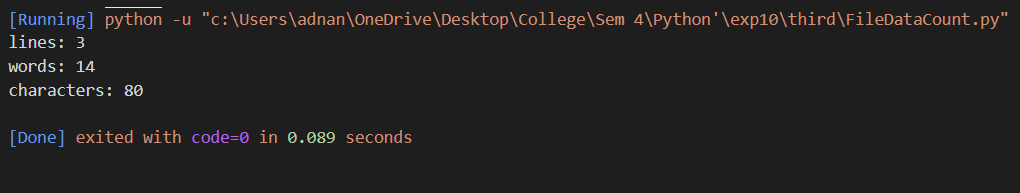
file.close()

print("lines:", number\_of\_lines)

print("words:", number\_of\_words)

print("characters:", number\_of\_characters)

Output:



**Content of a file:**

Hello, this is Adnan

Welcome to my woooooorld

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+")

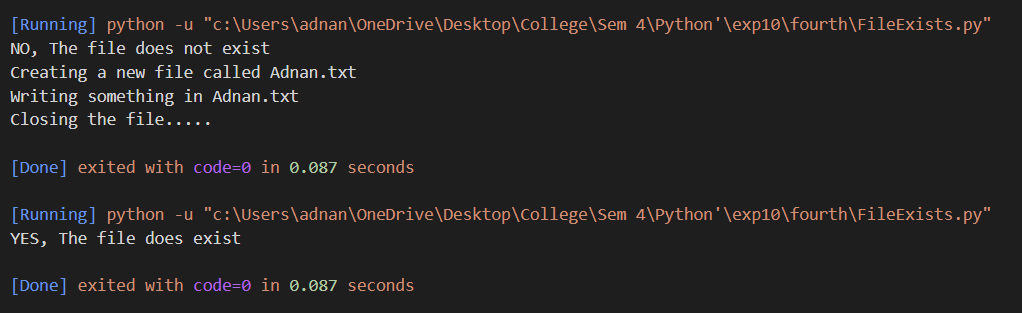
print("Writing something in Adnan.txt")

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

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

f1.close()

**Output**:



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

|  |  |  |
| --- | --- | --- |
| EXPERIMENT NO. 11 | | |
|  | | |
| Aim | : | Program to create a package and module for data structures like  stack and queues. |
| 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 quiet 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 | : | From this experiment we have studied built in functions like pop(), append() used to perform operations on data structures like stack and queue.  By Student - Adnan Shaikh |

**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")

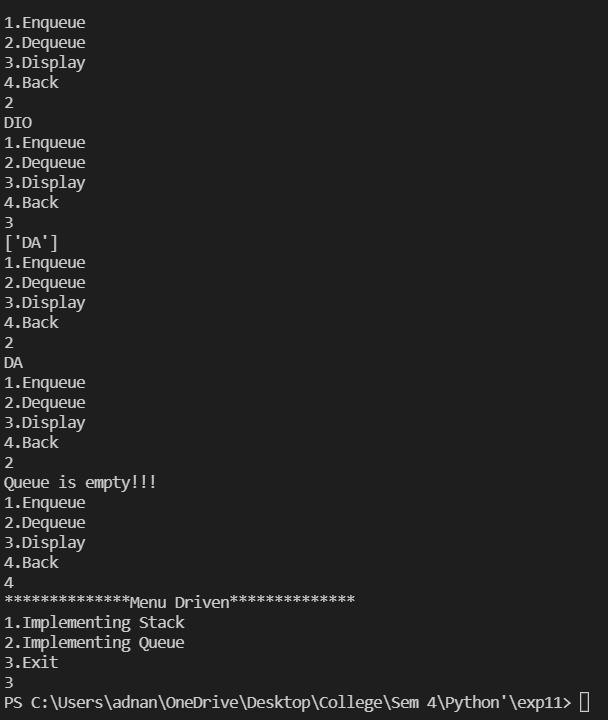
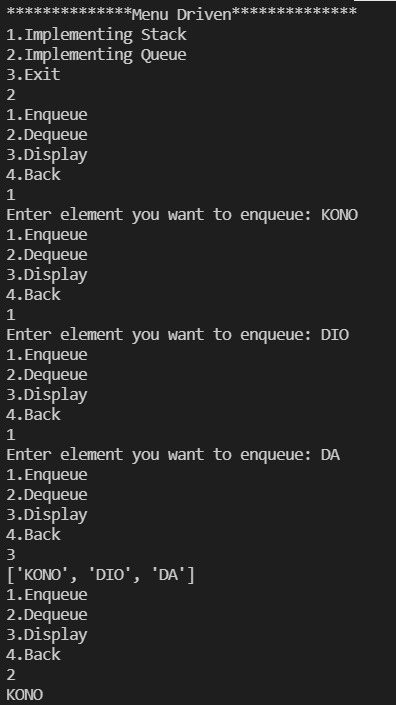
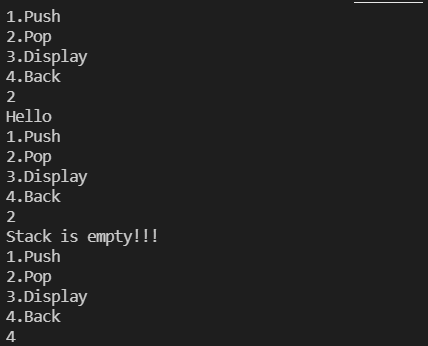
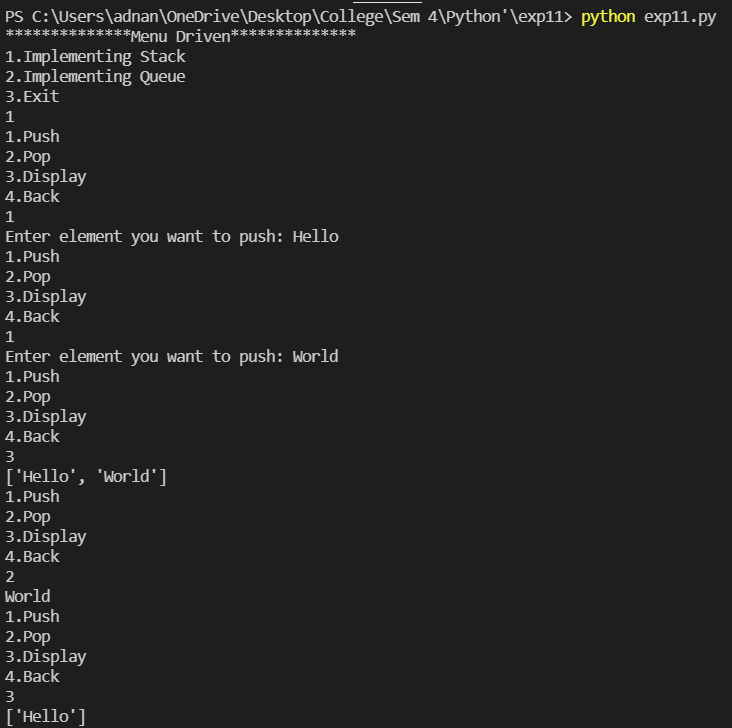
elif num == 3:

break

else:

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

**Output**:



|  |  |  |
| --- | --- | --- |
| EXPERIMENT NO. 12 | | |
|  | | |
| Aim | : | Program to create simple socket for basic info 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.recv() | This method  message | receives | TCP |
| s.send() | This method  message | transmits | TCP |
| s.recvfrom() | This method  message | receives | UDP |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | 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  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 tenet 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.  By Student Adnan Shaikh | | | |

**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))

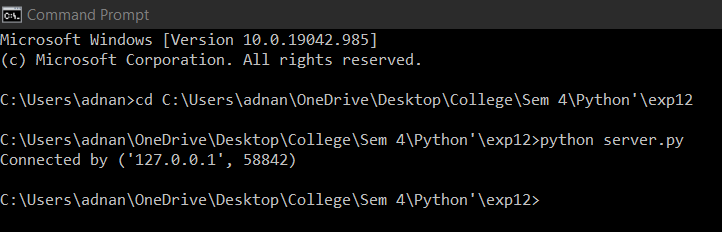
s.sendall(b'Hello, world')

data = s.recv(1024)

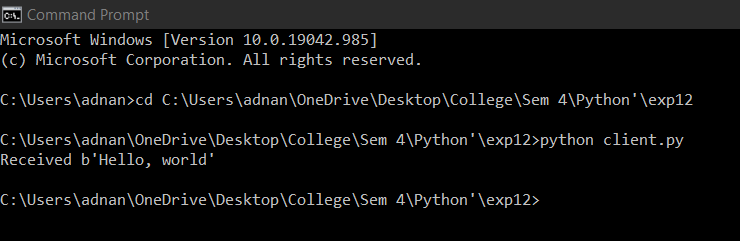
print('Received', repr(data))

**Output**:

Server Side:



Client Side:



|  |  |  |
| --- | --- | --- |
| EXPERIMENT NO. 13 | | |
|  | | |
| Aim | : | Demonstrate Database connectivity |
| 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-pythonorPython -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 anywheredb\_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. By Student Adnan Shaikh |

**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**:

