Experiment No 1: in lab manual

Experiment no 2:
1. Exploring basics of python like data types (strings, list, array, dictionaries, set, tuples) and control statements
(i) Numbers

```
a = 5
print(a, "is of type", type(a))

a = 2.0
print(a, "is of type", type(a))

a = 1+2j
print(a, "is complex number?", isinstance(1+2j,complex))
```

(ii) Python List

```
a = [5,10,15,20,25,30,35,40]

# a[2] = 15
print("a[2] = ", a[2])

# a[0:3] = [5, 10, 15]
print("a[0:3] = ", a[0:3])

# a[5:] = [30, 35, 40]
print("a[5:] = ", a[5:])
```

(iii) Python Tuple

```
t = (5,'program', 1+3j)

# t[1] = 'program'
print("t[1] = ", t[1])

# t[0:3] = (5, 'program', (1+3j))
print("t[0:3] = ", t[0:3])

# Generates error
# Tuples are immutable
```

```python
t[0] = 10
```

(iv) Python Strings

```python
s = "This is a string"
print(s)
s = '''A multiline
string'''
print(s)
```

```python
s = 'Hello world!'

# s[4] = 'o'
print("s[4] = ", s[4])

# s[6:11] = 'world'
print("s[6:11] = ", s[6:11])

# Generates error
# Strings are immutable in Python
s[5] ='d'
```

(v) Python Set

```python
a = {5,2,3,1,4}

# printing set variable
print("a = ", a)

# data type of variable a
print(type(a))
```

(vi) Python Dictionary

```
d = {1:'value','key':2}
print(type(d))

print("d[1] = ", d[1]);

print("d['key'] = ", d['key']);

# Generates error
print("d[2] = ", d[2]);
```

(vii) Conversion between data types

We can convert between different data types by using different type conversion functions like `int()`, `float()`, `str()`, etc.

```
>>> float(5)
5.0
```

Conversion from float to int will truncate the value (make it closer to zero).

```
>>> int(10.6)
10
>>> int(-10.6)
-10
```

Conversion to and from string must contain compatible values.

```
>>> float('2.5')
2.5
>>> str(25)
'25'
>>> int('1p')
Traceback (most recent call last):
  File "<string>", line 301, in runcode
  File "<interactive input>", line 1, in <module>
```

```
ValueError: invalid literal for int() with base 10: '1p'
```

We can even convert one sequence to another.

```
>>> set([1,2,3])
{1, 2, 3}
>>> tuple({5,6,7})
(5, 6, 7)
>>> list('hello')
['h', 'e', 'l', 'l', 'o']
```

To convert to dictionary, each element must be a pair:

```
>>> dict([[1,2],[3,4]])
{1: 2, 3: 4}
>>> dict([(3,26),(4,44)])
{3: 26, 4: 44}
```

Conditional Statements in Python

Syntax

```
if condition1:
    statements

elif condition2:
    statements

else:
    statements
```

Consider the example below:

```
X = 10
Y = 12

if X < Y: print('X is less than Y') elif X > Y:
    print('X is greater than Y')
else:
    print('X and Y are equal')
```

Output: *X is less than Y*

While Loop

Here, first the condition is checked and if it's true, control will move inside the loop and execute the statements inside the loop until the condition becomes false. We use this loop when we are not sure how many times we need to execute a group of statements or you can say that when we are unsure about the number of iterations.
Consider the example:
Syntax and Usage

```
count = 0
while (count < 10):
  print ( count )
  count = count + 1

print ("Good bye!")
```

Output = *0*

```
1
2
3
4
5
6
7
8
9
Good bye!
```

For Loop
Like the While loop, the For loop also allows a code block to be repeated certain number of times. The difference is, in For loop we know the amount of iterations required unlike While loop, where iterations depends on the condition. You will get a better idea about the difference between the two by looking at the syntax:
Syntax

```
for variable in Sequence:
    statements
```

Notice here, we have specified the range, that means we know the number of times the code block will be executed.
Consider the example:

```
fruits = ['Banana', 'Apple',  'Grapes']

for index in range(len(fruits)):
  print (fruits[index])
```

Output: *Banana   Apple   Grapes*

Nested Loops
It basically means a loop inside a loop. It can be a For loop inside a While loop and vice-versa. Even a For loop can be inside a For loop or a While loop inside a While loop.
Consider the example:

```
count = 1
```

```
for i in range(10):
    print (str(i) * i)

    for j in range(0, i):
        count = count +1
```

Output

```
1
22
333
4444
55555
666666
7777777
88888888
999999999
```

Experiment No 3.

Creating functions, classes and objects using python. Demonstrate exception handling and inheritance.

--Code for Class Object and Function

```python
class Person:
    "This is a person class"
    age = 10

    def greet(self):
        print('Hello')


# create a new object of Person class
harry = Person()

# Output: <function Person.greet>
print(Person.greet)

# Output: <bound method Person.greet of <__main__.Person object>>
print(harry.greet)

# Calling object's greet() method
# Output: Hello
harry.greet()
```

Demonstrating exception handling

```python
# import module sys to get the type of exception
```

```
import sys

randomList = ['a', 0, 2]

for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except:
        print("Oops!", sys.exc_info()[0], "occurred.")
        print("Next entry.")
        print()
print("The reciprocal of", entry, "is", r)
```

Experiment no 4

4.a. Python program to append data to existing file and then display the entire file

Code:

```
def file_read(fname):
        from itertools import islice
        with open(fname, "w") as myfile:
                myfile.write("Python Exercises\n")
                myfile.write("Java Exercises")
        txt = open(fname)
        print(txt.read())
file_read('abc.txt')
```

4.b. Python program to count number of lines, words and characters in a file

Code:

```
fname = input("Enter file name: ")

num_words = 0

with open(fname, 'r') as f:
    for line in f:
        words = line.split()
        num_words += len(words)
print("Number of words:")
print(num_words)
```

4.c. Python program to display file available in current directory

Code:

```
import os

path ="C:/python"
#we shall store all the file names in this list
filelist = []

for root, dirs, files in os.walk(path):
       for file in files:
        #append the file name to the list
              filelist.append(os.path.join(root,file))

#print all the file names
for name in filelist:
    print(name)
```

Example : Get the list of all files with a specific extension

In this example, we will take a path of a directory and try to list all the files, with a specific extension **.py** here, in the directory and its sub-directories recursively.
Code:

```
import os

path ="C:\ambika"

for root, dirs, files in os.walk(path):
       for file in files:
              if(file.endswith(".py")):
                     print(os.path.join(root,file))
```

Experiment No 5

Creating GUI with python containing widgets such as labels, textbox, radio, checkboxes and custom dialog boxes

Code:

```
from tkinter import *

class MyWindow:
    def __init__(self, win):
        self.lbl1=Label(win, text='First number')
        self.lbl2=Label(win, text='Second number')
        self.lbl3=Label(win, text='Result')
        self.t1=Entry(bd=3)
        self.t2=Entry()
        self.t3=Entry()
        self.btn1 = Button(win, text='Add')
        self.btn2=Button(win, text='Subtract')
        self.lbl1.place(x=100, y=50)
        self.t1.place(x=200, y=50)
        self.lbl2.place(x=100, y=100)
        self.t2.place(x=200, y=100)
        self.b1=Button(win, text='Add', command=self.add)
        self.b2=Button(win, text='Subtract')
        self.b2.bind('<Button-1>', self.sub)
        self.b1.place(x=100, y=150)
        self.b2.place(x=200, y=150)
        self.lbl3.place(x=100, y=200)
        self.t3.place(x=200, y=200)
    def add(self):
        self.t3.delete(0, 'end')
        num1=int(self.t1.get())
        num2=int(self.t2.get())
        result=num1+num2
        self.t3.insert(END, str(result))
    def sub(self, event):
        self.t3.delete(0, 'end')
        num1=int(self.t1.get())
        num2=int(self.t2.get())
        result=num1-num2
        self.t3.insert(END, str(result))

window=Tk()
mywin=MyWindow(window)
window.title('Hello Python')
window.geometry("400x300+10+10")
window.mainloop()
```

****

```
#!/usr/bin/python3


from tkinter import *
```

```
class Checkbar(Frame):
   def __init__(self, parent=None, picks=[], side=LEFT, anchor=W):
      Frame.__init__(self, parent)
      self.vars = []
      for pick in picks:
         var = IntVar()
         chk = Checkbutton(self, text=pick, variable=var)
         chk.pack(side=side, anchor=anchor, expand=YES)
         self.vars.append(var)
   def state(self):
      return map((lambda var: var.get()), self.vars)
if __name__ == '__main__':
   root = Tk()
   lng = Checkbar(root, ['Python', 'Ruby', 'Perl', 'C++'])
   tgl = Checkbar(root, ['English','German'])
   lng.pack(side=TOP,  fill=X)
   tgl.pack(side=LEFT)
   lng.config(relief=GROOVE, bd=2)

   def allstates():
      print(list(lng.state()), list(tgl.state()))
   Button(root, text='Quit', command=root.quit).pack(side=RIGHT)
   Button(root, text='Peek', command=allstates).pack(side=RIGHT)
   root.mainloop()
```

Experiment no 6

Menu driven program for data structure using built in function for link list, stack and queue.

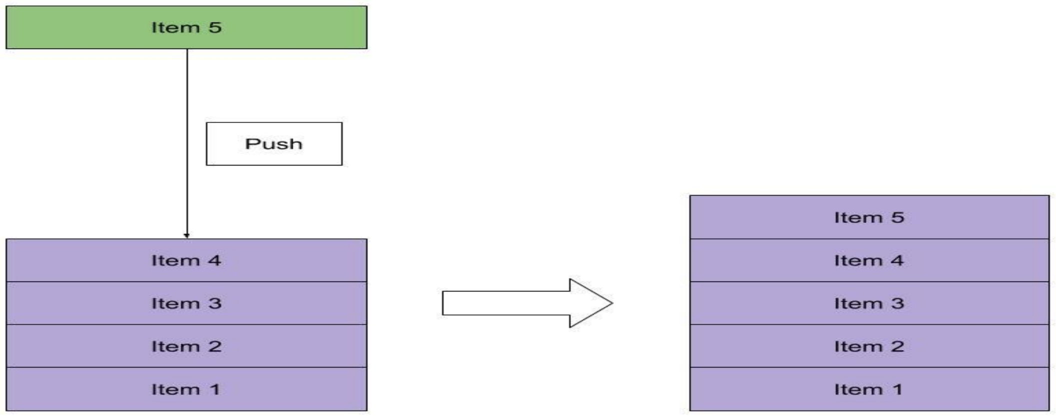
How do they Work?

Stack

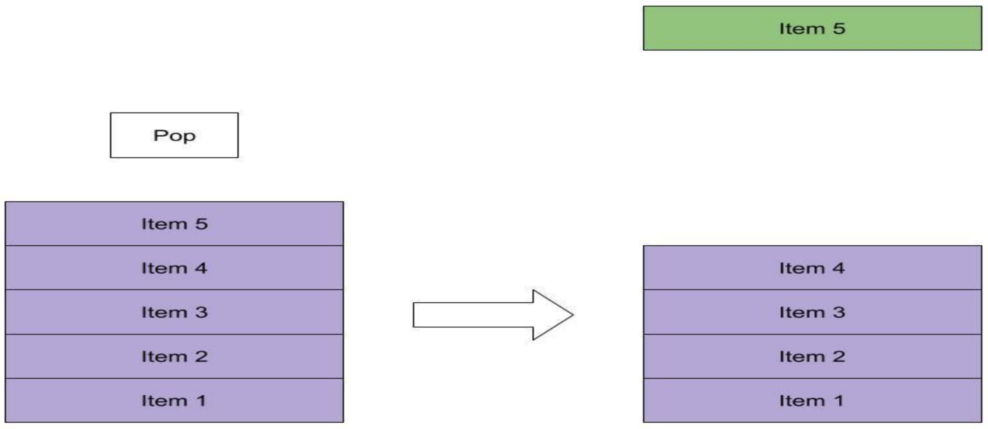
Stacks, like the name suggests, follow the **Last-in-First-Out** (LIFO) principle. As if stacking coins one on top of the other, the last coin we put on the top is the one that is the first to be removed from the stack later.

To implement a stack, therefore, we need two simple operations:

- push - adds an element to the top of the stack:



- pop - removes the element at the top of the stack:



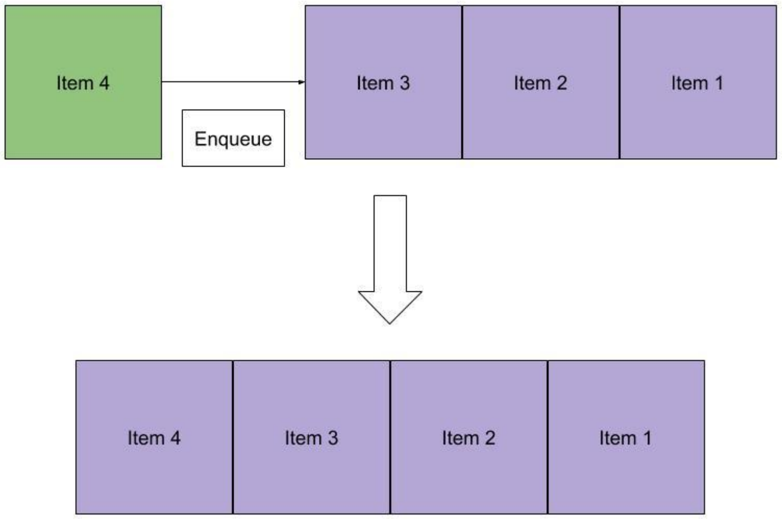
```
letters = []


# Let's push some letters into our list

letters.append('c')
letters.append('a')

letters.append('t')

letters.append('g')


# Now let's pop letters, we should get 'g'

last_item = letters.pop()

print(last_item)


# If we pop again we'll get 't'

last_item = letters.pop()

print(last_item)


# 'c' and 'a' remain

print(letters) # ['c', 'a']
```

Queue

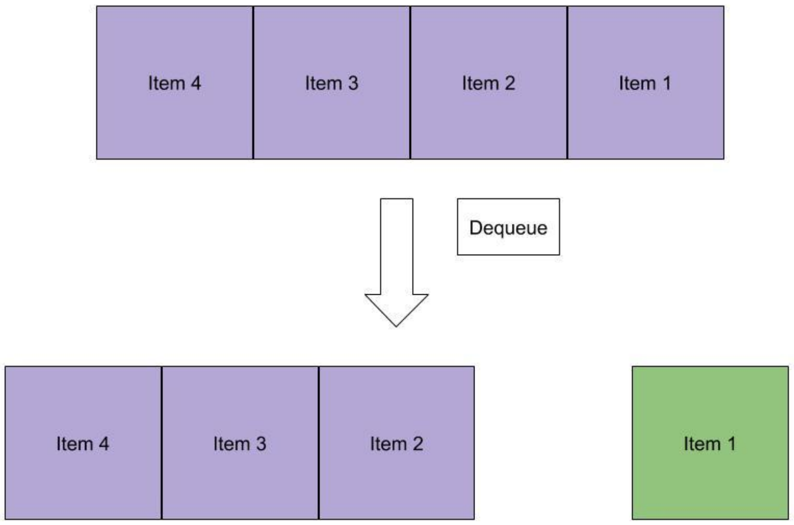
Queues, like the name suggests, follow the **First-in-First-Out** (FIFO) principle. As if waiting in a queue for the movie tickets, the first one to stand in line is the first one to buy a ticket and enjoy the movie.

To implement a queue, therefore, we need two simple operations:

- `enqueue` - adds an element to the end of the queue:



- `dequeue` - removes the element at the beginning of the queue:



fruits = []

```
# Let's enqueue some fruits into our list

fruits.append('banana')

fruits.append('grapes')

fruits.append('mango')

fruits.append('orange')


# Now let's dequeue our fruits, we should get 'banana'

first_item = fruits.pop(0)

print(first_item)


# If we dequeue again we'll get 'grapes'

first_item = fruits.pop(0)

print(first_item)


# 'mango' and 'orange' remain

print(fruits) # ['c', 'a']
```

Experiment No 7

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

```
"""
        importent links:
        http://sebastianraschka.com/Articles/2014_sqlite_in_python_tutorial.html
        http://www.pythoncentral.io/introduction-to-sqlite-in-python/
```

```
"""


import sqlite3
import subprocess as sp

"""



database code



"""




def create_table():
       conn = sqlite3.connect('testdb2.sqlite')

       cursor = conn.cursor()

       query = '''
           CREATE TABLE IF NOT EXISTS student(
              id INTEGER PRIMARY KEY,
              roll INTEGER,
              name TEXT,
               phone TEXT
           )
       '''

       cursor.execute(query)

       conn.commit()
       conn.close()




def add_student(roll,name,phone):
       conn = sqlite3.connect('testdb2.sqlite')

       cursor = conn.cursor()
```

```
        query = '''
            INSERT INTO student( roll, name, phone )
                         VALUES ( ?,?,? )
        '''

        cursor.execute(query,(roll,name,phone))

        conn.commit()
        conn.close()



def get_students():
        conn = sqlite3.connect('testdb2.sqlite')

        cursor = conn.cursor()

        query = '''
            SELECT roll, name, phone
            FROM student
        '''

        cursor.execute(query)
        all_rows = cursor.fetchall()

        conn.commit()
        conn.close()

        return all_rows

def get_student_by_roll(roll):
        conn = sqlite3.connect('testdb2.sqlite')

        cursor = conn.cursor()

        query = '''
            SELECT roll, name, phone
            FROM student
            WHERE roll = {}
        ''' .format(roll)

        cursor.execute(query)
        all_rows = cursor.fetchall()

        conn.commit()
        conn.close()

        return all_rows

def update_student(roll,name,phone):
        conn = sqlite3.connect('testdb2.sqlite')

        cursor = conn.cursor()

        query = '''
            UPDATE student
            SET name = ?, phone = ?
            WHERE roll = ?
        '''

        cursor.execute(query,(name,phone,roll))

        conn.commit()
        conn.close()


def delete_student(roll):
        conn = sqlite3.connect('testdb2.sqlite')

        cursor = conn.cursor()

        query = '''
            DELETE
            FROM student
            WHERE roll = {}
        ''' .format(roll)

        cursor.execute(query)
        all_rows = cursor.fetchall()

        conn.commit()
        conn.close()

        return all_rows
```

```
create_table()


"""


main code


"""



def add_data(id_,name,phone):
        add_student(id_,name,phone)
def get_data():
        return get_students()

def show_data():
        students = get_data()
        for student in students:
                print(student)

def show_data_by_id(id_):
        students = get_student_by_roll(id_)
        if not students:
                print("No data found at roll",id_)
        else:
                print (students)

def select():
        sp.call('clear',shell=True)
        sel = input("1. Add data\n2.Show
Data\n3.Search\n4.Update\n5.Delete\n6.Exit\n\n")


        if sel=='1':
                sp.call('clear',shell=True)
                id_ = int(input('id: '))
                name = input('Name: ')
                phone = input('phone: ')
                add_data(id_,name,phone)
        elif sel=='2':
                sp.call('clear',shell=True)
                show_data()
                input("\n\npress enter to back:")
        elif sel=='3':
                sp.call('clear',shell=True)
                id__ = int(input('Enter Id: '))
                show_data_by_id(id__)
                input("\n\npress enter to back:")
        elif sel=='4':
                sp.call('clear',shell=True)
                id__ = int(input('Enter Id: '))
                show_data_by_id(id__)
                print()
                name = input('Name: ')
                phone = input('phone: ')
                update_student(id__,name,phone)
                input("\n\nYour data has been updated \npress enter to back:")
        elif sel=='5':
                sp.call('clear',shell=True)
                id__ = int(input('Enter Id: '))
                show_data_by_id(id__)
                delete_student(id__)
                input("\n\nYour data has been deleted \npress enter to back:")
        else:
                return 0;
        return 1;


while(select()):
        pass
```

Experiment No 8:

Included in lab manual

Experiment No 9:

Included in lab manual

Experiment No 10:

Programs on Threading using python.

Code:

Thread.py

```
import thread # import the thread module
import time # import time module

def cal_sqre(num): # define the cal_sqre function
    print(" Calculate the square root of the given number")
    for n in num:
        time.sleep(0.3) # at each iteration it waits for 0.3 time
        print(' Square is : ', n * n)

def cal_cube(num): # define the cal_cube() function
    print(" Calculate the cube of  the given number")
    for n in num:
        time.sleep(0.3) # at each iteration it waits for 0.3 time
        print(" Cube is : ", n * n *n)

arr = [4, 5, 6, 7, 2] # given array

t1 = time.time() # get total time to execute the functions
cal_sqre(arr) # call cal_sqre() function
cal_cube(arr) # call cal_cube() function
```

```

print(" Total time taken by threads is :", time.time() - t1) # print the total time
```

Output:

```
Calculate the square root of the given number
 Square is:  16
 Square is:  25
 Square is:  36
 Square is:  49
 Square is:  4
 Calculate the cube of the given number
 Cube is:  64
 Cube is:  125
 Cube is:  216
 Cube is:  343
 Cube is:  8
 Total time taken by threads is: 3.005793809890747
```

write a program to use the threading module in Python Multithreading.

Threading.py

```
import time # import time module
import threading
from threading import *
def cal_sqre(num): # define a square calculating function
    print(" Calculate the square root of the given number")
    for n in num: # Use for loop
        time.sleep(0.3) # at each iteration it waits for 0.3 time
        print(' Square is : ', n * n)

def cal_cube(num): # define a cube calculating function
```

```
    print(" Calculate the cube of  the given number")
    for n in num: # for loop
        time.sleep(0.3) # at each iteration it waits for 0.3 time
        print(" Cube is : ", n * n *n)

ar = [4, 5, 6, 7, 2] # given array

t = time.time() # get total time to execute the functions
#cal_cube(ar)
#cal_sqre(ar)
th1 = threading.Thread(target=cal_sqre, args=(ar, ))
th2 = threading.Thread(target=cal_cube, args=(ar, ))
th1.start()
th2.start()
th1.join()
th2.join()
print(" Total time taking by threads is :", time.time() - t) # print the total time
print(" Again executing the main thread")
print(" Thread 1 and Thread 2 have finished their execution.")
```

Output:

```
Calculate the square root of the given number
 Calculate the cube of the given number
 Square is:  16
 Cube is:  64
 Square is:  25
 Cube is:  125
 Square is:  36
 Cube is:  216
 Square is:  49
 Cube is:  343
```

```
Square is:  4
Cube is:  8
Total time taken by threads is: 1.5140972137451172
Again executing the main thread
Thread 1 and Thread 2 have finished their execution.
```

Experiment No 11:

Included in lab manual

Experiment No 12:

Included in lab manual