1A

```
marks1=int(input("Enter Test 1 Marks:"))
marks2=int(input("Enter Test 2 Marks:"))
marks3=int(input("Enter Test 3 Marks:"))
minimum=min(marks1,marks2,marks3)
sumofbest2=marks1+marks2+marks3-minimum
avgofbest2=sumofbest2/2
print("Average of best Two :",avgofbest2)
```

1B

```
val = int(input("Enter a value : "))
str_val = str(val)
if str_val == str_val[::-1]:
    print("Entered Value is Palindrome")
else:
    print("Entered Value is Not a Palindrome")
for i in range(10):
  if str_val.count(str(i)) > 0:
      print(str(i), "appears", str_val.count(str(i)), "times")
```

2A

```
def fn(n):
 if n == 1:
    return 0
 elif n == 2:
    return 1
 else:
    return fn(n - 1) + fn(n - 2)
num = int(input("Enter a Number:"))
if num > 0:
 print("fn(", num, ") = ", fn(num), sep="")
else:
 print("Error in input")
```

2B

```
def BinToDec(b):
 return int(b, 2)
print("Enter the Binary Number: ", end="")
bnum = input()
dnum = BinToDec(bnum)
print("\nEquivalent Decimal Value = ", dnum)
def OctToHex(o):
 return hex(int(o, 8))
print("Enter Octal Number: ", end="")
onum = input()
hnum = OctToHex(onum)
print("\nEquivalent Hexadecimal Value =", hnum[2:].upper())
```

3A

```
s = input("Enter a sentence: ")
w, d, u, l = 0, 0, 0, 0
l_w = s.split()
w = len(l_w)
for c in s:
   if c.isdigit():
     d = d + 1
   elif c.isupper():
     u = u + 1
   elif c.islower():
     l = l + 1
print ("No of Words: ", w)
print ("No of Digits: ", d)
print ("No of Uppercase letters: ", u)
print ("No of Lowercase letters: ", l)
```

3B

```
str1 = input("Enter First String:\n")
str2 = input("Enter Second String\n")
if len(str2) < len(str1):
 short = len(str2)
 long = len(str1)
else:
 short = len(str1)
 long = len(str2)
matchCnt = 0
for i in range(short):
 if str1[i] == str2[i]:
  matchCnt += 1
print("Similarity between two said String:")
print(matchCnt/ long)
```

4A INSERTION SORT

```
def insertion_sort(alist):

    for i in range(1,len(alist)):
        temp = alist[i]
        j = i - 1
        while (j >= 0 and temp <alist[j]):
          alist[j + 1] = alist[j]
          j = j - 1
        alist[j + 1] = temp

alist = input('Enter The List of Numbers:').split()
alist = [int(x) for x in alist]

insertion_sort(alist)
print('Sorted List: ', end='')
print(alist)
```

MERGE SORT

4B

MERGE SORT

```
def mergesort(list1):
    if len(list1) > 1:
       mid = len(list1) // 2
       left = list1[:mid]
       right = list1[mid:]
       mergesort(left)
       mergesort(right)
       i = 0
       j = 0
       k = 0
       while i < len(left) and j < len(right):
         if left[i] < right[j]:
            list1[k] = left[i]
            i = i + 1
            k = k + 1
         else:
            list1[k] = right[j]
            j = j + 1
            k = k + 1
       while i < len(left):
         list1[k] = left[i]
         i = i + 1
         k = k + 1
       while j < len(right):
         list1[k] = right[j]
         j = j + 1
         k = k + 1
list1 = input('enter the list of values to be sorted: ').split()
list1 = [int(x) for x in list1]
mergesort(list1)
print(list1)
```

```
class sol_Roman:
  def roman_to_integerNo(self, s):
      roman_no = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}
      integer_no = 0
      for i in range(len(s)):
          if i > 0 and roman_no[s[i]] > roman_no[s[i - 1]]:
             integer_no += roman_no[s[i]] - 2 * roman_no[s[i - 1]]
      else:
        integer_no += roman_no[s[i]]
      return integer_no
print("Roman Numerical to Integer is:",
sol_Roman().roman_to_integerNo(input("Enter the Roman Numericals:")))
```

5A

```
import re

def isphonenumber(numStr):
  if len(numStr) != 12:
      return False
  for i in range(len(numStr)):
    if i == 3 or i == 7:
        if numStr[i] != "-":
            return False
    else:
        if numStr[i].isdigit() == False:
            return False
  return True

def chkphonenumber(numStr):
   ph_no_pattern = re.compile(r'^\d{3}-\d{3}-\d{4}$')
   if ph_no_pattern.match(numStr):
      return True
   else:
      return False

ph_num = input("Enter a phone number : ")
print("Without using Regular Expression")
if isphonenumber(ph_num):
   print("Valid phone number")
else:
   print("Invalid phone number")

print("Using Regular Expression")
if chkphonenumber(ph_num):
    print("Valid phone number")
else:
    print("Invalid phone number")
```

5B(CREATE TEXT FILE)

```
import re

phone_regex = re.compile(r'\+\d{12}')
email_regex = re.compile(r'[A-Za-z0-9._]+@[A-Za-z0-9]+\.[A-Z|a-z]{2,}')
with open('pattern.txt', 'r') as f:

    for line in f:

            matches = phone_regex.findall(line)

            for match in matches:
                print(match)
            matches = email_regex.findall(line)

            for match in matches:
                        print(match)
```

6A(CREATE FILE)

```
import os.path

import sys

fname = input("Enter the filename :")
if not os.path.isfile(fname):
    print("File", fname, "doesn't exists")
    sys.exit(0)


infile = open(fname, "r")

lineList = infile.readlines()

for i in range(20):

   print(i + 1, ":", lineList[i])
word = input("Enter a word : ")
cnt = 0

for line in lineList:

   cnt += line.count(word)

print("The word", word, "appears", cnt, "times in the file")
```

6B(ZIP FILE)

```
import os
import sys
import pathlib
import zipfile

dirName = input("Enter Directory name that you want to backup : ")

if not os.path.isdir(dirName):
    print("Directory", dirName, "doesn't exists")
    sys.exit(0)

curDirectory = pathlib.Path(dirName)

with zipfile.ZipFile("myzip.zip", mode="w") as archive:
    for file_path in curDirectory.rglob("*"):
        archive.write(file_path, arcname=file_path.relative_to(curDirectory))

if os.path.isfile("myzip.zip"):
    print("Archive", "myzip.zip", "created successfully")
else:
    print("Error in creating zip archive")
```

7A

```
import math
class Shape:
 def __init__(self):
      self.area = 0
      self.name = ""
 def showArea(self):
    print("The area of the", self.name, "is", self.area, "units")
class Circle(Shape):
 def __init__(self, radius):
    self.area = 0
    self.name = "Circle"
    self.radius = radius
 def calcArea(self):
    self.area = math.pi * self.radius * self.radius
class Rectangle(Shape):
 def __init__(self, length, breadth):
   self.area = 0
   self.name = "Rectangle"
   self.length = length
   self.breadth = breadth
 def calcArea(self):
   self.area = self.length * self.breadth
class Triangle(Shape):
 def __init__(self, base, height):
     self.area = 0
     self.name = "Triangle"
     self.base = base
     self.height = height
 def calcArea(self):
    self.area = self.base * self.height / 2
c1 = Circle(5)
c1.calcArea()
c1.showArea()
r1 = Rectangle(5, 4)
r1.calcArea()
r1.showArea()
t1 = Triangle(3, 4)
t1.calcArea()
t1.showArea()
```

7B

```
class Employee:
  def __init__(self):
     self.name = ""
     self.empId = ""
     self.dept = ""
     self.salary = 0
  def getEmpDetails(self):
     self.name = input("Enter Employee name : ")
     self.empId = input("Enter Employee ID : ")
     self.dept = input("Enter Employee Dept : ")
     self.salary = int(input("Enter Employee Salary : "))
  def showEmpDetails(self):
     print("Employee Details")
     print("Name : ", self.name)
     print("ID : ", self.empId)
     print("Dept : ", self.dept)
     print("Salary : ", self.salary)
  def updtSalary(self):
     self.salary = int(input("Enter new Salary : "))
     print("Updated Salary", self.salary)
e1 = Employee()
e1.getEmpDetails()
e1.showEmpDetails()
e1.updtSalary()
```

8A

```
class PaliStr:
    def __init__(self):
        self.isPali = False

    def chkPalindrome(self, myStr):
        if myStr == myStr[::-1]:
            self.isPali = True
        else:
            self.isPali = False
        return self.isPali

class PaliInt(PaliStr):
    def __init__(self):
        self.isPali = False


    def chkPalindrome(self, val):
        temp = val
        rev = 0
        while temp != 0:
            dig = temp % 10
            rev = (rev*10) + dig
            temp = temp // 10

        if val == rev:
            self.isPali = True
        else:
            self.isPali = False
        return self.isPali

st = input("Enter a string :")
stObj = PaliStr()
if stObj.chkPalindrome(st):
    print("Given string is a Palindrome")
else:
    print("Given string is not a Palindrome")

val = int(input("Enter a integer : "))

intObj = PaliInt()
if intObj.chkPalindrome(val):
    print("Given integer is a Palindrome")
else:
    print("Given integer is not a Palindrome")
```

9A

```
import requests
import os
from bs4 import BeautifulSoup


url = 'https://xkcd.com/1/'


if not os.path.exists('xkcd_comics'):
    os.makedirs('xkcd_comics')


while True:

    res = requests.get(url)
    res.raise_for_status()


    soup = BeautifulSoup(res.text, 'html.parser')


    comic_elem = soup.select('#comic img')
    if comic_elem == []:
        print('Could not find comic image.')
    else:
        comic_url = 'https:' + comic_elem[0].get('src')


        print(f'Downloading {comic_url}...')
        res = requests.get(comic_url)
        res.raise_for_status()


        image_file = open(os.path.join('xkcd_comics', os.path.basename(comic_url)), 'wb')
        for chunk in res.iter_content(100000):
            image_file.write(chunk)
        image_file.close()

    prev_link = soup.select('a[rel="prev"]')[0]
    if not prev_link:
        break
    url = 'https://xkcd.com' + prev_link.get('href')

print('All comics downloaded.')
```

9B

```
from openpyxl import Workbook
from openpyxl.styles import Font

wb = Workbook()
sheet = wb.active
sheet.title = "Language"
wb.create_sheet(title = "Capital")

lang = ["Kannada", "Telugu", "Tamil"]
state = ["Karnataka", "Telangana", "Tamil Nadu"]
capital = ["Bengaluru", "Hyderabad", "Chennai"]
code =['KA', 'TS', 'TN']

sheet.cell(row = 1, column = 1).value = "State"
sheet.cell(row = 1, column = 2).value = "Language"
sheet.cell(row = 1, column = 3).value = "Code"


ft = Font(bold=True)
for row in sheet["A1:C1"]:
    for cell in row:
        cell.font = ft

for i in range(2,5):
    sheet.cell(row = i, column = 1).value = state[i-2]
    sheet.cell(row = i, column = 2).value = lang[i-2]
    sheet.cell(row = i, column = 3).value = code[i-2]

wb.save("demo.xlsx")


sheet = wb["Capital"]

sheet.cell(row = 1, column = 1).value = "State"
sheet.cell(row = 1, column = 2).value = "Capital"
sheet.cell(row = 1, column = 3).value = "Code"

ft = Font(bold=True)
for row in sheet["A1:C1"]:
    for cell in row:
        cell.font = ft

for i in range(2,5):
    sheet.cell(row = i, column = 1).value = state[i-2]
    sheet.cell(row = i, column = 2).value = capital[i-2]
    sheet.cell(row = i, column = 3).value = code[i-2]

wb.save("demo.xlsx")

srchCode = input("Enter state code for finding capital ")
for i in range(2,5):
    data = sheet.cell(row = i, column = 3).value
    if data == srchCode:
        print("Corresponding capital for code", srchCode, "is", sheet.cell(row = i, column = 2).val


sheet = wb["Language"]
```