1a)

```
m1 = int(input("Enter marks for test1 : "))
m2 = int(input("Enter marks for test2 : "))
m3 = int(input("Enter marks for test3 : "))
if m1 <= m2 and m1 <= m3:
   avgMarks = (m2 + m3) / 2
elif m2 <= m1 and m2 <= m3:
   avgMarks = (m1 + m3) / 2
elif m3 <= m1 and m2 <= m2:
   avgMarks = (m1 + m2) / 2
print("Average of best two test marks  out of three test's marks is", avgMarks);
```

1b)

```
val = int(input("Enter a value : "))
str_val = str(val)
if str_val == str_val[::-1]:
   print("Palindrome")
else:
   print("Not 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)
def OctToHex(o):
   return hex(int(o, 8))
print("Enter the Binary Number: ", end="")
bnum = input()
dnum = BinToDec(bnum)
print("\nEquivalent Decimal Value = ", dnum)
```

```
print("Enter Octal Number: ", end="")
onum = input()
hnum = OctToHex(onum)
print("\nEquivalent Hexadecimal Value =", hnum[2:].upper())
```

3a)

```
sentence = input("Enter a sentence : ")
wordList = sentence.split(" ")
print("This sentence has", len(wordList), "words")
digCnt = upCnt = loCnt = 0
for ch in sentence:
   if '0' <= ch <= '9':
      digCnt += 1
   elif 'A' <= ch <= 'Z':
      upCnt += 1
   elif 'a' <= ch <= 'z':
      loCnt += 1
print("This sentence has", digCnt, "digits", upCnt, "upper case letters", loCnt, "lower case letters")
```

3b)

```
import difflib
def string_similarity(str1, str2):
    result =  difflib.SequenceMatcher(a=str1.lower(), b=str2.lower())
    return result.ratio()
str1 = 'Python Exercises'
str2 = 'Python Exercises'
print("Original string:")
print(str1)
print(str2)
print("Similarity between two said strings:")
print(string_similarity(str1,str2))
```

4a)

```
import random

def merge_sort(lst):

   if len(lst) > 1:

      mid = len(lst) // 2

      left_half = lst[:mid]

      right_half = lst[mid:]

      merge_sort(left_half)

      merge_sort(right_half)
```

```
        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                lst[k] = left_half[i]
                i += 1
            else:
                lst[k] = right_half[j]
                j += 1
            k += 1
        while i < len(left_half):
            lst[k] = left_half[i]
            i += 1
            k += 1
        while j < len(right_half):
            lst[k] = right_half[j]
            j += 1
            k += 1
    return lst
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
my_list = []
for i in range(10):
    my_list.append(random.randint(0, 999))
print("\nUnsorted List")
print(my_list)
```

```
print("Sorting using Insertion Sort", insertion_sort(my_list))
my_list = []
for i in range(10):
    my_list.append(random.randint(0, 999))


print("\nUnsorted List")
print(my_list)
print("Sorting using Merge Sort", merge_sort(my_list))
```

4b)

```
def roman2Dec(romStr):
    roman_dict ={'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}
    romanBack = list(romStr)[::-1]
    value = 0
    rightVal = roman_dict[romanBack[0]]
    for numeral in romanBack:
        leftVal = roman_dict[numeral]
        if leftVal < rightVal:
            value -= leftVal
        else:
            value += leftVal
        rightVal = leftVal
    return value
romanStr = input("Enter a Roman Number : ")
print(roman2Dec(romanStr))
```

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)
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('example.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)

```
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)

```
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 area(self):
        pass
class Triangle(Shape):
    def __init__(self, base, height):
        self.base = base
        self.height = height
    def area(self):
        return 0.5 * self.base * self.height
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return math.pi * self.radius ** 2
class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width
    def area(self):
        return self.length * self.width
triangle = Triangle(4, 5)
circle = Circle(3)
```

```
rectangle = Rectangle(6, 8)
print("Area of the triangle:", triangle.area())
print("Area of the circle:", circle.area())
print("Area of the rectangle:", rectangle.area())
```

7b)

```
class Employee:
    def __init__(self, name, employee_id, department, salary):
        self.name = name
        self.employee_id = employee_id
        self.department = department
        self.salary = salary
    def update_salary_by_department(self, department, new_salary):
        if self.department == department:
            self.salary = new_salary
employee1 = Employee("John Doe", 1001, "HR", 50000)
employee2 = Employee("Jane Smith", 1002, "Finance", 60000)
employee3 = Employee("Michael Johnson", 1003, "HR", 55000)
print("Original Salaries:")
print(f"{employee1.name} - {employee1.salary}")
print(f"{employee2.name} - {employee2.salary}")
print(f"{employee3.name} - {employee3.salary}")
department_to_update = "HR"
new_salary_for_hr = 60000
employee1.update_salary_by_department(department_to_update, new_salary_for_hr)
employee3.update_salary_by_department(department_to_update, new_salary_for_hr)
print("\nUpdated Salaries:")
print(f"{employee1.name} - {employee1.salary}")
print(f"{employee2.name} - {employee2.salary}")
print(f"{employee3.name} - {employee3.salary}")
```

8)

```
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)

```
source_file = "source.xlsx"
source_workbook = openpyxl.load_workbook(source_file)
source_sheet = source_workbook.active
destination_file = "destination.xlsx"
destination_workbook = openpyxl.Workbook()
destination_sheet = destination_workbook.active
for row in source_sheet.iter_rows(values_only=True):
    destination_sheet.append(row)
destination_workbook.save(destination_file)
source_workbook.close()
destination_workbook.close()
print("Data has been copied from source to destination.")
```

10a)

```
from PyPDF2 import PdfWriter, PdfReader
num = int(input("Enter page number you want combine from multiple documents "))
pdf1 = open('AAT-1.pdf', 'rb')
pdf2 = open('AAD QB1.pdf', 'rb')
pdf_writer = PdfWriter()
pdf1_reader = PdfReader(pdf1)
page = pdf1_reader.pages[num - 1]
pdf_writer.add_page(page)
pdf2_reader = PdfReader(pdf2)
page = pdf2_reader.pages[num - 1]
pdf_writer.add_page(page)
```

```
with open('output.pdf', 'wb') as output:
    pdf_writer.write(output)
```

10b)

```
import json
with open("weather.json", "r") as json_file:
    weather_data = json.load(json_file)
location = weather_data["location"]
temperature = weather_data["temperature"]
humidity = weather_data["humidity"]
conditions = weather_data["conditions"]
print(f"Weather in {location}:")
print(f"Temperature: {temperature}°F")
print(f"Humidity: {humidity}%")
print(f"Conditions: {conditions}")
```