#program1a

s=input(**"enter a string"**)  
**if** s==s[::-1]:  
 print(**"enter the string is a palindrome"**)  
**else**:  
 print(**"enter the string is not palindrome"**)  
  
 l=list(s)  
 l.reverse()  
 reverse=**""**.join(l)  
 **if** s==reverse:  
 print(**"enter string is not palindrome"**)  
 **else**:  
 print(**"not a palindrome"**)  
  
c=s  
l=len(s)  
**for** i **in** s:  
 **if** i!=c[l-1]:  
 print(**"not a palindrome"**)  
 **break** l-=1  
 **else**:  
 print(**"entered string is palindrome"**)  
  
s=input(**"enter a string"**)  
lst=[]  
  
**for** word **in** s.split():  
 lst.append(word[0].upper()+word[1:])  
  
cam=**" "**.join(lst)  
print(**"string '%s' converted to camel case is '%s' "** %(s,cam))  
  
s=input(**"enter string:"**)  
v=0  
c=0  
**for** char **in** s:  
 **if** char **in**[**"a"**,**"e"**,**"i"**,**"o"**,**"u"**] **or** char **in**[**"A"**,**"E"**,**"I"**,**"O"**,**"U"**]:  
 v+=1  
 **elif** char.isalpha():  
 c+=1  
print(**"number of vowels is %d and numbers of cconstants is %d"** %(v,c))

#program1b

maxdays = [**None**, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]  
line = input(**"enter the date as dd/mm/yyyy : "**)  
(dd, mm, yy) = line.split(**'/'**)  
dd = int(dd)  
mm = int(mm)  
yy = int(yy)  
**if** (yy % 4 == 0 **and** yy % 100 != 0) **or** (yy % 400 == 0) :  
 maxdays[2] = 29  
**if** mm < 1 **or** mm > 12 :  
 print (**"invalid month"**)  
**elif** dd < 1 **or** dd > maxdays[mm] :  
 print (**"invalid date"**)  
**else**:  
 print (**"date ok"**)

#program2a

**def** find\_unique(\*all):  
 **for** word **in** all:  
 unique\_char\_list=list(set(word))  
 print(**"Unique characters in "**+word+**":"**+str(unique\_char\_list))  
  
find\_unique(**'aaaa'**, **'abcd'**, **'abba'**, **'xyz'**, **'abcba'**)

#program2b

print(**"Enter n:"**,end=**""**)  
n=int(input())  
pt = []  
**for** i **in** range(n) :  
 pt.append([])  
 pt[i].append(1)  
 **for** j **in** range(1, i) :  
 pt[i].append(pt[i-1][j-1] + pt[i-1][j])  
 **if** i != 0 : pt[i].append(1)

**for** i **in** range(n) :print( **" "** \* (n - i), end = **""**, sep = **""**)  
 **for** j **in** range(i + 1) :  
 print(**"{0:6}"**.format(pt[i][j]), end = **""**, sep = **""**)  
 print()

#program3a

**def** combine(\*all, init = **"result: "**, sep = **','**):  
 **return** init + sep.join(all)  
  
print(combine(**'this'**, **'is'**, **'a'**, **'test'**))  
print(combine(**'this'**, **'is'**, **'a'**, **'test'**, init = **'fool '**, sep= **' - '**))

#program3b

word\_dict = {}  
  
**def** create\_dict():  
 **global** word\_dict  
 word\_dict = {}  
 ch = **"y"  
 while** (ch == **"y"**) **or** (ch == **"Y"**):  
 print(**"\nEnter word:"**, end=**""**)  
 word = input()  
 print(**"\nEnter meaning:"**, end=**""**)  
 meaning = input()  
 word\_dict[word] = meaning  
 print(**"\nDo you want to continue adding words(y or n):"**, end=**""**)  
 ch = input()  
  
**def** add\_word():  
 **global** word\_dict  
 print(**"\nEnter word:"**, end=**""**)  
 word = input()  
 print(**"\nEnter meaning:"**, end=**""**)  
 meaning = input()  
 word\_dict[word] = meaning  
  
**def** find\_meaning(w):  
 **return** word\_dict[w]

**def** find\_word\_same\_meaning(mng):  
 words = []  
 **for** w, m **in** word\_dict.items():  
 **if** mng == m:  
 words.append(w)  
 **return** words  
  
**def** display\_sorted():  
 **for** w, m **in** word\_dict.items():  
 print(**"%s ==> %s"** % (w, m))  
 print(**"Sorted list of words : "**)  
 print(sorted(word\_dict.keys()))  
  
  
**def** main():  
 ch = **"y"  
 while** (ch == **"Y" or** ch == **"y"**):  
 print(**"1: Create new dictionary"**)  
 print(**"2: Add new word"**)  
 print(**"3: Find meaning"**)  
 print(**"4: Find word with same meaning"**)  
 print(**"5: Display sorted list of words"**)  
 print(**"6: Quit"**)  
 print(**"Enter Choice: "**, end=**""**)  
 option = int(input())  
 **if** option == 1:  
 create\_dict()  
 **elif** option == 2:  
 add\_word()  
 **elif** option == 3:  
 print(**"Enter word:"**, end=**""**)  
 word = input()  
 print(**"Meaning:%s"** % (find\_meaning(word)))  
 **elif** option == 4:  
 print(**"Enter meaning:"**, end=**""**)  
 meaning = input()  
 print(**"Words with same meaning:"**, end=**""**)  
 print(find\_word\_same\_meaning(meaning))  
 **elif** option == 5:  
 display\_sorted()  
 **elif** option == 6:  
 quit()  
  
 print(**"\nDo you want to continue(y or n)?"**, end=**""**)  
 ch = input()  
  
  
main()

#program4a

state\_dict = { }  
f = open(**"stateinfo.txt"**)  
  
**for** line **in** f :  
 line = line.strip()  
 (state, city) = line.split(**':'**)  
 **if** state **not in** state\_dict :  
 state\_dict[state] = open(state, **'w'**)  
 print(state, city, file = state\_dict[state])  
f.close()  
  
**for** fh **in** state\_dict.values() :  
 fh.close()

#program4b

**def** histogram(s):  
 d = dict()  
 **for** c **in** s:  
 d[c] = d.get(c,0) + 1  
 **return** d  
  
*# OR  
  
# def histogram(s):  
# d = dict()  
# for c in s:  
# if c not in d:  
# d[c] = 1  
# else:  
# d[c] = d[c] + 1  
# return d***def** print\_hist(h):  
 key\_list = sorted(h.keys())  
 **for** key **in** key\_list:  
 print(key, h.get(key))  
  
print\_hist(histogram(**'brontosaurus'**))

#program5a

**class** Node:  
 **def** \_\_init\_\_(self, cargo = **None**, next = **None**):  
 self.cargo = cargo  
 self.next = next  
  
 **def** \_\_str\_\_(self):  
 **return** str(self.cargo)  
  
  
**def** print\_list(node):  
 i = 0  
 **while** i < len(node):  
 print(node[i],)  
 node[i] = node[i].next  
 i+=1  
  
**def** link\_nodes(node):  
 i = 0  
 **while** (i < len(node)):  
**if** i < len(node)- 1:  
 node[i].next = node[i+1]  
 **else**:  
 node[i].next = **None** i += 1  
  
node = {}  
number\_Of\_Nodes = int(input(**'Enter the number of nodes to be creates'**))  
i=0  
**while** (i < number\_Of\_Nodes):  
 node\_Val = int(input(**'Enter the value for the node'**))  
 node[i] = Node(node\_Val)i+=1  
  
link\_nodes(node)  
print(**'The list of nodes created are'**)  
print\_list(node)

#program5b

**import** os  
  
*# Set the directory to start from*print(**"Enter path to traverse :"**, end=**""**)  
rootDir = input()  
**if** (os.path.exists(rootDir)):  
 dir\_count = 0  
 file\_count = 0  
 **for** dirName, subdirList, fileList **in** os.walk(rootDir):  
 print(**'Found directory: %s'** % dirName)  
 *# check to ignore starting directory while taking directory count  
 # normpath returns the normalized path eliminating double slashes etc.* **if** os.path.normpath(rootDir) != os.path.normpath(dirName):  
 dir\_count += 1  
 **for** fname **in** fileList:  
 file\_count += 1  
 print(**'\t%s'** % fname)  
  
 print(**"No: of subdirectories :"**, dir\_count, end=**""**)  
 print(**"\nNo: of files :"**, file\_count, end=**""**)  
**else**:  
 print(**"Entered path doesn't exist"**)

#program6a

**class** StackFull(Exception) :  
 **def** \_\_init\_\_(self) :  
 self.msg = **'stack is full'  
 def** \_\_str\_\_(self) :  
 **return** self.msg  
  
**class** StackEmpty(Exception) :  
 **def** \_\_init\_\_(self) :  
 self.msg = **'stack is empty'  
 def** \_\_str\_\_(self) :  
 **return** self.msg  
  
  
**class** MyStack :  
 *# assumed that the size is not negative* **def** \_\_init\_\_(self, size = 10) :  
 self.mylist = [ ]  
 self.size = size  
 **def** push(self, elem) :  
 l = len(self.mylist)  
 **if** l < self.size :  
 self.mylist.append(elem)  
 **else**:  
 **raise** StackFull()  
 **def** pop(self) :  
 **if** len(self.mylist) == 0 :  
 **raise** StackEmpty()  
 **else**:  
 **return** self.mylist.pop()  
  
s = MyStack(3)  
*# what follows could be menu driven***try**:  
 s.push(11)  
 s.push(22)  
 s.push(33)  
*# s.push(44)* print(s.pop())  
 print(s.pop())  
 print(s.pop())  
 print(s.pop())  
**except** Exception **as** e :  
 print(e)

#program6b

**class** Place:  
 **def** \_\_init\_\_(self, city, \*places):  
 self.city = city  
 self.places = list(places)  
  
 **def** add(self, place):  
 self.places.append(place)  
  
 **def** remove(self, place):  
 *# exception not checked* self.places.remove(place)  
  
 **def** disp(self):  
 print(self.city)  
 **for** place **in** self.places:  
 print(**"\t"**, place)  
  
  
p = Place(**'mysore'**, **'chamundi hills'**, **'zoo'**)  
p.disp()  
p.add(**'krs'**)  
p.disp()  
p.remove(**'zoo'**)  
p.disp()

#program7a

**import** re  
  
*#Find all phone numbers having 4 consecutive 0s at the end.*f = open(**"details.txt"**,**"r"**)  
print(**"\n2a Solution\n"**)  
**for** line **in** f:  
 m=re.search(**r"[a-zA-z]+\s+(\d{2,3}-\d{4}0{4})\s+"**,line)  
 **if** m:  
 print(m.group(1))  
f.close()  
  
*#Find all names having phone numbers with 3 digit area code.*f = open(**"details.txt"**,**"r"**)  
print(**"\n2b Solution\n"**)  
**for** line **in** f:  
 m=re.search(**r"([a-zA-z]+)\s+\d{3}-\d{8}\s+"**,line)  
 **if** m:  
 print(m.group(1))  
f.close()  
  
*#Find the total number of people having Gmail id.*f = open(**"details.txt"**,**"r"**)  
all\_lines = f.read()  
print(**"\n2c Solution\n"**)  
L = re.findall(**r"\w+@gmail\.com"**,all\_lines)  
print(L)  
print(len(L))  
f.close()  
*#Find user name part of email id for all people whose name start with 'G' or 'E' and ends with 'y'*f = open(**"details.txt"**,**"r"**)  
print(**"\n2d Solution\n"**)  
**for** line **in** f:  
 m = re.search(**r"^[GE][a-z]\*y\s+.\*\s+(\w+)@\w+\.\w+"**,line)  
 **if** m:  
 print(m.group(1))  
f.close()  
  
*#Find all names whose phone numbers are not in proper format.*f = open(**"details.txt"**,**"r"**)  
print(**"\n2e Solution\n"**)  
**for** line **in** f:  
 m = re.search(**r".\*\s+\d{2,3}-\d{8}"**,line)  
 **if not** m:  
 m=re.search(**r"(^[A-Z][a-z]+)"**,line)  
 print(m.group(1))  
f.close()

#program7b

**import** re  
  
line = **'''this String is a multiline string  
used to test the usage of re.multilinestring in a  
multiline string'''***#To search the word "string" in line*match\_Obj = re.finditer(**r"\bstring\b"**,line,re.I)  
  
  
**'''If the word to be searched is stored in the variable "word\_to\_find", then the  
 regular expression can be written as follows  
  
word\_to\_find = "string"  
match\_Obj = re.finditer(r"\b%s\b"%word\_to\_find,line,re.I) '''  
  
  
for** word **in** match\_Obj:  
 print(word.group()+**" at index "**,int(word.start()))  
  
**import** re  
  
line = **'this is a line of text !'**L=re.findall(**r"[^aeiou \t]"**,line)  
print(L)  
  
**import** re  
  
L = [**"apple"**,**"4sdj"**,**"\_5dfkjghd"**,**"\_\_next"**,**"abcd"**,**"02352"**]  
  
**for** item **in** L:  
 **if** re.search(**r"^[\d\_]"**,item):  
 print(item)

#program8a

**import** re  
  
f = open(**"sample.txt"**,**"r"**)  
str = f.read()  
f.close()  
  
**def** change\_upper\_start(m):  
 **return** m.group(1).upper()  
  
**def** change\_upper\_startline(m):  
 **return** m.group(1)+m.group(2).upper()  
*#Remove spaces at the beginning and convert first char to uppercase*s1 = re.sub(**r"^\s\*([a-z])"**,change\_upper\_start,str)  
  
*#Insert whitespace at the end of each sentence*s2 = re.sub(**"([.?!])"**,**r"\1 "**,s1)  
  
*#Remove extra spaces between words*s3= re.sub(**r"[ \t]+"**,**" "**,s2)  
  
*#Convert first char of each sentence to uppercase*s4=re.sub(**r"([.?!]\s+)([a-z])"**,change\_upper\_startline,s3)  
  
*#Remove consecutive duplicate words*s5=re.sub(**r"(\b\w+\b\s+)(\1)+"**,**r"\1"**,s4)  
  
f=open(**"converted.txt"**,**"w"**)  
f.write(s5)  
f.close()

#program8b

nterms = int(input(**'How many terms?'**))  
  
n1 = 0  
n2 = 1  
count = 2  
  
*#check if the number of terms are valid***if** nterms <=0:  
 print(**'Please enter a positive number'**)  
**elif** nterms == 1:  
 print(**'Fibonacci sequence'**)  
 print(**'1'**)  
 print(**'\n'**)  
**else**:  
 print(**"Fibonacci sequence"**)  
 print(n1)  
 print(n2)  
 **while** count < nterms:  
 nth = n1 + n2  
 print(nth,)  
 n1 = n2  
 n2 = nth  
 count += 1