# **INDEX**

1. Demonstrate the working of ‘id’ and ‘type’ functions

2. To find all prime numbers within a given range.

3. To print ‘n terms of Fibonacci series using iteration.

4. To demonstrate use of slicing in string

5.

a. To add 'ing' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add 'ly' instead. If the string length of the given string is less than 3, leave it unchanged. Sample String : 'abc' Expected Result : 'abcing' Sample String : 'string' Expected Result : 'stringly'

b. To get a string from a given string where all occurrences of its first char have been changed to '$', except the first char itself.

6.

a. To compute the frequency of the words from the input. The output should output after sorting the key alphanumerically.

b. Write a program that accepts a comma separated sequence of words as input and prints the words in a comma-separated sequence after sorting them alphabetically.

7. Write a program that accepts a sequence of whitespace separated words as input and prints the words after removing all duplicate words and sorting them alphanumerically.

8. To demonstrate use of list & related functions

9. To demonstrate use of Dictionary& related functions

10. To demonstrate use of tuple, set& related functions

11. To implement stack using list

12. To implement queue using list

13. To read and write from a file

14. To copy a file

15. To demonstrate working of classes and objects

16. To demonstrate class method & static method

17. To demonstrate constructors

18. To demonstrate inheritance

19. To demonstrate aggregation/composition

# 1. Demonstrate the working of ‘id’ and ‘type’ functions

a = 2

b = 3

c = 2

print(id(a))

print(id(b))

print(id(c))

#a and c have same id

#type is used to check object's class/type

if type(a) == int:

print("a is int type")

'''

-------------------Output-------------------------

1599103232

1599103264

1599103232

a is int type

---------------------EOF--------------------------

'''

# 2. To find all prime numbers within a given range.

def isPrime(n):

for i in range(2,n):

if n % i == 0:

return False

return True

start = int(input("Enter start: "))

end = int(input("Enter end: "))

for x in range(start,end + 1):

if isPrime(x):

print(x)

'''

-------------------Output-------------------------

Enter start: 100

Enter end: 200

101

103

107

109

113

127

131

137

139

149

151

157

163

167

173

179

181

191

193

197

199

---------------------EOF--------------------------

'''

# 3. To print ‘n terms of Fibonacci series using iteration.

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

a= 0

b=1

for x in range(n):

a,b = b,a+b

print(b)

'''

-------------------Output-------------------------

1

2

3

5

8

13

21

34

55

89

---------------------EOF--------------------------

'''

# 4. To demonstrate use of slicing in string

a = "ankur"

print(a[0]) #accessing single character

print(a[-1]) #last character

print(a[0:3]) #slicing

print(a[3:]) #upto end

'''

-------------------Output-------------------------

a

r

ank

ur

---------------------EOF--------------------------

'''

# 5.

# a. To add 'ing' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add 'ly' instead. If the string length of the given string is less than 3, leave it unchanged. Sample String : 'abc' Expected Result : 'abcing' Sample String : 'string' Expected Result : 'stringly'

# b. To get a string from a given string where all occurrences of its first char have been changed to '$', except the first char itself.

def ingly(s):

if str(s).endswith("ing"):

return str(s) + 'ly'

else:

return s + 'ing'

def dolloring(s):

x = s.split(" ")

for i in range(len(x)):

c = len(x[i]) - 1

x[i] = x[i][0] + "$" \* c

s = ""

for y in x:

s+=y + " "

return s

a = "abc"

b = "inkling"

c = "the quick brown fox"

print(ingly(a))

print(ingly(b))

print(dolloring(c))

'''

-------------------Output-------------------------

abcing

inklingly

t$$ q$$$$ b$$$$ f$$

---------------------EOF--------------------------

'''

# 6.

# a. To compute the frequency of the words from the input. The output should output after sorting the key alphanumerically.

# b. Write a program that accepts a comma separated sequence of words as input and prints the words in a comma-separated sequence after sorting them alphabetically.

def sorting(s):

x = str(s).split(",")

x.sort()

s = ''

for y in x:

s+=y + ','

s = s.rstrip(',')

return s

def freqcount(s):

x = str(s).split(' ')

d = dict()

for y in x:

if y in d:

d[y]+=1

else:

d[y] = 1

return d

a = 'yo yo honey singh'

b = 'the,quick,brown,fox'

c = freqcount(a)

for x in sorted(c):

print(x,':',c[x])

print(sorting(b))

'''

-------------------Output-------------------------

honey : 1

singh : 1

yo : 2

brown,fox,quick,the

---------------------EOF--------------------------

'''

# 7. Write a program that accepts a sequence of whitespace separated words as input and prints the words after removing all duplicate words and sorting them alphanumerically.

a = input("Enter string: ")

a = a.split(' ')

x = []

for i in a:

if i in x:

continue

x.append(i)

for i in sorted(x):

print(i,end=' ')

'''

-------------------Output-------------------------

Enter string: the the quick brown fox the

brown fox quick the

---------------------EOF--------------------------

'''

# 8. To demonstrate use of list & related functions

a = list()

b = []

c = [1,'a',True]

a.append(1)

# a = [1]

c.count(1)

# 1

c.extend(a)

# c = [1,'a',True,1]

c.sort()

# c = [1,1,'a',True]

c.pop()

# c = [1,1,'a']

c.insert(2,'x')

# c = [1,1,'x','a']

c.clear()

# c = []

# 9. To demonstrate use of Dictionary& related functions

dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}

dict.get('Name')

# 'Manni'

dict.keys()

# ['Name', 'Age']

dict.items()

# [('Name', 'Manni'), ('Age', 7)]

dict.values()

# ['Manni', 7]

# 10. To demonstrate use of tuple, set& related functions

tup = ('physics', 'chemistry', 1997, 2000)

x = [1,2,3]

tuple(x)

# (1,2,3)

tup.count(item)

tup.index(item)

len(tup)

# 4

s = {10,20,10,30,20}

print(set) # {10,20,30} #no repeated value

# 11. To implement stack using list

stack = ["Amar", "Akbar", "Anthony"]

stack.append("Ram")

stack.append("Iqbal")

print(stack)

print(stack.pop())

print(stack)

print(stack.pop())

print(stack)

'''

-------------------Output-------------------------

['Amar', 'Akbar', 'Anthony', 'Ram', 'Iqbal']

Iqbal

['Amar', 'Akbar', 'Anthony', 'Ram']

Ram

['Amar', 'Akbar', 'Anthony']

---------------------EOF--------------------------

'''

# 12. To implement queue using list

stack = ["Amar", "Akbar", "Anthony"]

stack.append("Ram")

stack.append("Iqbal")

print(stack)

print(stack.pop(0))

print(stack)

print(stack.pop(0))

print(stack)

'''

-------------------Output-------------------------

['Amar', 'Akbar', 'Anthony', 'Ram', 'Iqbal']

Amar

['Akbar', 'Anthony', 'Ram', 'Iqbal']

Akbar

['Anthony', 'Ram', 'Iqbal']

---------------------EOF--------------------------

'''

# 13. To read and write from a file

f = open('TextFile1.txt','r')

c = f.read()

print(c)

f.close()

f = open('TextFile1.txt','w')

f.write(c)

f.flush()

f.close()

'''

-------------------Output-------------------------

hello

how are you?

---------------------EOF--------------------------

'''

# 14. To copy a file

o = open('originalFilePath.anyExt','r')

d = open('destinationFilePath.anyExt','w')

d.write(o.read())

o.close()

d.close()

# 15. To demonstrate working of classes and objects

class class1:

'''documentation'''

#attributes and methods

pass

#object initialization

a = class1()

a.\_\_setattr\_\_('name','test')

#test

print(a.\_\_getattribute\_\_('name'))

# 16. To demonstrate class method & static method and

# 17. To demonstrate constructors

class student:

'''documentation'''

totalStd = 0 # class attribute

#constructor

def \_\_init\_\_(self,name,marks=0):

self.name = name #object attributes

self.\_\_marks = marks #private object attribute

student.totalStd += 1

#class methods

def calculateGrade(self):

if self.\_\_marks > 80:

return 'A'

if self.\_\_marks > 60:

return 'B'

if self.\_\_marks > 40:

return 'C'

return 'D'

#static method

@staticmethod

def getTotalStudents():

return student.totalStd

a = student('ankur')

b = student('ankur',88)

a.totalStd=10

print(a.totalStd) # 10

print(student.getTotalStudents()) # 2

print(a.calculateGrade()) # D

print(b.calculateGrade()) # A

# 18. To demonstrate inheritance

class User:

name = ""

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

self.name = name

def printName(self):

print ("Name = " + self.name)

class Programmer(User):

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

self.name = name

def doPython(self):

print ("Programming Python")

brian = User("brian")

brian.printName()

diana = Programmer("Diana")

diana.printName()

diana.doPython()

'''

-------------------Output-------------------------

Name = brian

Name = Diana

Programming Python

---------------------EOF--------------------------

'''

# 19. To demonstrate aggregation/composition

#composition (composed at class level)

class Salary:

def \_\_init\_\_(self,pay):

self.pay=pay

def get\_total(self):

return (self.pay\*12)

class Employee:

def \_\_init\_\_(self,pay,bonus):

self.pay=pay

self.bonus=bonus

self.obj\_salary=Salary(self.pay)

def annual\_salary(self):

return "Total: " + str(self.obj\_salary.get\_total()+self.bonus)

obj\_emp=Employee(100,10)

print (obj\_emp.annual\_salary()) #1210

#aggregation (at object level)

class Salary:

def \_\_init\_\_(self,pay):

self.pay=pay

def get\_total(self):

return (self.pay\*12)

class Employee:

def \_\_init\_\_(self,pay,bonus):

self.pay=pay

self.bonus=bonus

def annual\_salary(self):

return "Total: " + str(self.pay.get\_total()+self.bonus)

obj\_sal=Salary(100)

obj\_emp=Employee(obj\_sal,10)

print (obj\_emp.annual\_salary()) #1210