3. Write a function that draws a Pyramid with # symbols

#

# # #

# # # # #

# # # # # # #

# Python3 code to demonstrate star pattern

# function to print spaces

def print\_space(space):

    # base case

    if (space == 0):

        return;

    print(" ", end = "");

    # recursively calling print\_space()

    print\_space(space - 1);

# function to print asterisks

def print\_asterisk(asterisk):

    # base case

    if(asterisk == 0):

        return;

    print("\* ", end = "");

    # recursively calling asterisk()

    print\_asterisk(asterisk - 1);

# function to print the pattern

def pattern(n, num):

    # base case

    if (n == 0):

        return;

    print\_space(n - 1);

    print\_asterisk(num - n + 1);

    print("");

    # recursively calling pattern()

    pattern(n - 1, num);

  # Driver Code

n = 5;

pattern(n, n);

4. Using turtles concept draw a wheel of your choice

# Python program to draw hexagon

# using Turtle Programming

import turtle

polygon = turtle.Turtle()

num\_sides = 6

side\_length = 70

angle = 360.0 / num\_sides

for i in range(num\_sides):

    polygon.forward(side\_length)

    polygon.right(angle)

turtle.done()

5. Write a program that draws Archimedean Spiral

Aim:

Program:

from turtle import \*

from math import \*

color ('blue')

down()

for i in range(200):

t= i/20 \* pi

x=(1 + 5\* t)\* cos (t)

y=(1 + 5 \* t)\* sin (t)

goto(x, y)

up()

done()

7. The time module provides a function, also named time that returns the current Greenwich Mean Time in “the epoch”, which is an arbitrary time used as a reference point. On UNIX systems, the epoch is 1 January 1970.

>>> import time

>>> time.time()

1437746094.5735958

Write a script that reads the current time and converts it to a time of day in hours, minutes, and seconds, plus the number of days since the epoch.

Ans:

import time

epoch = time.time()

seconds\_in\_a\_day = 24 \* 60 \* 60

seconds\_in\_an\_hour = 60 \* 60

seconds\_in\_a\_minute = 60

days = epoch // seconds\_in\_a\_day

hours = (epoch % seconds\_in\_a\_day) // seconds\_in\_an\_hour + 8

minutes = (epoch % seconds\_in\_a\_day) % seconds\_in\_an\_hour // seconds\_in\_a\_minute

seconds = (epoch % seconds\_in\_a\_day) % seconds\_in\_an\_hour % seconds\_in\_a\_minute

print("%s: %s: %s: %s" %(days, hours, minutes, seconds))

8. Given n+r+1 <= 2r . n is the input and r is to be determined. Write a program which computes minimum value of r that satisfies the above.

Ans:

n=int(input("N="))

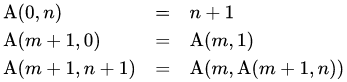
for r in range(10):

if (n+r+1)<=(2\*\*r):

print("minimum values of r that satisfies the above expression is:",r)

9. Write a program that evaluates Ackermann function.

In [computability theory](https://en.wikipedia.org/wiki/Computability_theory), the **Ackermann function**, named after [Wilhelm Ackermann](https://en.wikipedia.org/wiki/Wilhelm_Ackermann), is one of the simplest[[1]](https://en.wikipedia.org/wiki/Ackermann_function#cite_note-FOOTNOTEMoninHinchey200361-1) and earliest-discovered examples of a [total](https://en.wikipedia.org/wiki/Total_function) [computable function](https://en.wikipedia.org/wiki/Computable_function) that is not [primitive recursive](https://en.wikipedia.org/wiki/Primitive_recursive_function). All primitive recursive functions are total and computable, but the Ackermann function illustrates that not all total computable functions are primitive recursive. After Ackermann's publication[[2]](https://en.wikipedia.org/wiki/Ackermann_function#cite_note-FOOTNOTEAckermann1928-2) of his function (which had three nonnegative integer arguments), many authors modified it to suit various purposes, so that today "the Ackermann function" may refer to any of numerous variants of the original function. One common version, the two-argument **Ackermann–Péter function**, is defined as follows for nonnegative integers *m* and *n*:

{\displaystyle {\begin{array}{lcl}\operatorname {A} (0,n)&=&n+1\\\operatorname {A} (m+1,0)&=&\operatorname {A} (m,1)\\\operatorname {A} (m+1,n+1)&=&\operatorname {A} (m,\operatorname {A} (m+1,n))\end{array}}} 

Program:

def ackermann(m,n):

if m == 0:

return (n + 1)

elif n == 0:

return ackermann(m - 1, 1)

else:

return ackermann(m - 1, ackermann(m, n - 1))

# Writte in next prompt

x=int(input("What is the value for m? "))

print(x)

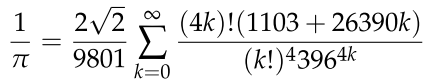
y=int(input("What is the value for n? "))

print(y)

print( "\nThe result of your inputs according to the Ackermann Function is:")

ackermann(x, y)

10. The mathematician Srinivasa Ramanujan found an infinite series that can be used to generate a numerical approximation of 1/ π : Write a function called estimate\_pi that uses this formula to compute and return an estimate of π.



It should use a while loop to compute terms of the summation until the last term is smaller than 1e-15 (which is Python notation for 10 -15). You can check the result by comparing it to math.pi.

Program:

def estimatePi():

import math

def factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n-1)

k=0

final=0

pi=0

while pi<1e-15:

a=factorial(4\*k)

b=(1103+26390\*k)

c=factorial(k)

d=c\*\*4

e=396\*\*(4\*k)

f=2\*math.sqrt(2)/9801

final+=(a\*b\*f)/(d\*e)

k+=1

pi=1/final

return pi

🡪 estimatePi()

11. Choose any five built-in string functions of C language. Implement them on your own in Python. You should not use string related Python built-in functions.

Program:

s=str(input())

con=s+s

dub=s\*4

cha=s[1:2]

news='@'+s[1:]

count=0

for letter in s:

count=count+1

print("Given String is: ",s)

print("concatenation of strings is: ",con)

print("doublicating the sting: ",dub)

print("Finding the Character using index in string:",cha)

print("Adding a Character to string: ",news)

print("counting the length of a string: ",count)

12. Given a text of characters, Write a program which counts number of vowels, consonants and special characters.

Program:

def countCharacterType(str):

vowels = 0

consonant = 0

specialChar = 0

digit = 0

# str.length() function to count

# number of character in given string.

for i in range(0, len(str)):

ch = str[i]

if ( (ch >= 'a' and ch <= 'z') or (ch >= 'A' and ch <= 'Z') ):

# To handle upper case letters

ch = ch.lower()

if (ch == 'a' or ch == 'e' or ch == 'i' or ch == 'o' or ch == 'u'):

vowels += 1

else:

consonant += 1

elif (ch >= '0' and ch <= '9'):

digit += 1

else:

specialChar += 1

print("Vowels:", vowels)

print("Consonant:", consonant)

print("Digit:", digit)

print("Special Character:", specialChar)

* str = "sjcet for sjcet@121"

countCharacterType(str)

13. Given a word which is a string of characters. Given an integer say ‘n’, Rotate each character by ‘n’ positions and print it. Note that ‘n’ can be positive or negative.

Program:

def rotate(word,n):

# slice string in two parts for left and right

Lfirst = word[0 : n]

Lsecond = word[n :]

Rfirst = word[0 : len(word)-n]

Rsecond = word[len(word)-n : ]

# now concatenate two parts together

print ("Left Rotation : ", (Lsecond + Lfirst) )

print ("Right Rotation : ", (Rsecond + Rfirst))

* w="pythonprogramming"
* rotate(w,5)

14. Given rows of text, write it in the form of columns.

Program:

row='st.johns College Of Engineering & Technology'

col={}

for cha in row:

if cha not in col:

col[cha]=1

else:

col[cha]=col[cha]+1

print(col)

15. Given a page of text. Count the number of occurrences of each latter (Assume case insensitivity and don’t consider special characters). Draw a histogram to represent the same.

Aim:

Program:

row='St jhons college of engineering and technology'

count={}

for cha in row:

if(cha!='@'or cha!='$'or cha!='&'):

cha=cha.lower()

if cha not in count:

count[cha]=1

else:

count[cha]+=1

print(count)

18. Write a program that reads a file, breaks each line into words, strips whitespace and punctuation from the words, and converts them to lowercase.

**Aim:**

**Text file:**

First we create a text file of which we want to count the words. Let this file be **sample.txt** with the following contents:

Mango banana apple pear

Banana grapes strawberry

Apple pear mango banana

Kiwi apple mango strawberry

**Note:** Make sure the text file is in same directory as the Python file.

**Program:**

# Open the file in read mode

text = open("sample.txt", "r")

# Create an empty dictionary

d = dict()

# Loop through each line of the file

for line in text:

# Remove the leading spaces and newline character

line = line.strip()

# Convert the characters in line to

# lowercase to avoid case mismatch

line = line.lower()

# Split the line into words

words = line.split(" ")

# Iterate over each word in line

for word in words:

# Check if the word is already in dictionary

if word in d:

# Increment count of word by 1

d[word] = d[word] + 1

else:

# Add the word to dictionary with count 1

d[word] = 1

# Print the contents of dictionary

for key in list(d.keys()):

print(key, ":", d[key])

19. Go to Project Gutenberg (http://gutenberg.org) and download your favorite out-of-copyright book in plain text format. Read the book you downloaded, skip over the header information at the beginning of the file, and process the rest of the words as before. Then modify the program to count the total number of words in the book, and the number of times each word is used. Print the number of different words used in the book. Compare different books by different authors, written in different eras.

Aim:

Program:

import string, time

def del\_punctuation(item):

'''

This function deletes punctuation from a word.

'''

punctuation = string.punctuation

for c in item:

if c in punctuation:

item = item.replace(c, '')

return item

def break\_into\_words():

'''

This function reads file, breaks it into

a list of used words in lower case.

'''

book = open('newbook.txt','r')

words\_list = []

for line in book:

for item in line.split():

item = del\_punctuation(item)

item = item.lower()

#print(item)

words\_list.append(item)

return words\_list

def create\_dict():

'''

This function calculates words frequency and

returns it as a dictionary.

'''

words\_list = break\_into\_words()

dictionary = {}

for word in words\_list:

if word not in dictionary:

dictionary[word] = 1

else:

dictionary[word] += 1

return dictionary

dictionary = create\_dict()

dictionary.pop(' ', None) # accidentally 5 empty strings appeared in the dictionary. why?

start\_time = time.time()

print('The total number of words in the book is {}'.format(len(break\_into\_words())))

print('The number of different words used in the book {}'.format(len(dictionary)))

function\_time = time.time() - start\_time

print('Running time is {0:.4f} s'.format(function\_time))

21. Consider all the files on your PC. Write a program which checks for duplicate files in your PC and displays their location. Hint: If two files have the same checksum, they probably have the same contents.

Aim:

Program:

import os

import hashlib

from collections import defaultdict

import csv

src\_folder = "../../"

def generate\_md5(fname, chunk\_size=1024):

"""

Function which takes a file name and returns md5 checksum of the file

"""

hash = hashlib.md5()

with open(fname, "rb") as f:

# Read the 1st block of the file

chunk = f.read(chunk\_size)

# Keep reading the file until the end and update hash

while chunk:

hash.update(chunk)

chunk = f.read(chunk\_size)

# Return the hex checksum

return hash.hexdigest()

if \_\_name\_\_ == "\_\_main\_\_":

"""

Starting block of script

"""

# The dict will have a list as values

md5\_dict = defaultdict(list)

file\_types\_inscope = ["ppt", "pptx", "pdf", "txt", "html",

"mp4", "jpg", "png", "xls", "xlsx", "xml", "vsd", "py", "json"]

# Walk through all files and folders within directory

for path, dirs, files in os.walk(src\_folder):

print("Analyzing {}".format(path))

for each\_file in files:

if each\_file.split(".")[-1].lower() in file\_types\_inscope:

# The path variable gets updated for each subfolder

file\_path = os.path.join(os.path.abspath(path), each\_file)

# If there are more files with same checksum append to list

md5\_dict[generate\_md5(file\_path)].append(file\_path)

# Identify keys (checksum) having more than one values (file names)

duplicate\_files = (

val for key, val in md5\_dict.items() if len(val) > 1)

# Write the list of duplicate files to csv file

with open("duplicates.csv", "w") as log:

# Lineterminator added for windows as it inserts blank rows otherwise

csv\_writer = csv.writer(log, quoting=csv.QUOTE\_MINIMAL, delimiter=",",

lineterminator="\n")

header = ["File Names"]

csv\_writer.writerow(header)

for file\_name in duplicate\_files:

csv\_writer.writerow(file\_name)

print("Done")

22. Consider turtle object. Write functions to draw triangle, rectangle, polygon, circle and sphere. Use object oriented approach.

Aim:

Program:

import turtle

import time

trtl=turtle.Turtle()

trtl.pencolor('red') #making colour of the pen red

trtl.pensize(5) #choosing the size of pen nib

#trtl.speed(1) #choosing the speed of drawing

trtl.shape('turtle') #choosing the shape of pen n

n=3 #starting for a triangle

shapes=['Triangle','Square','Pentagon','Hexagon','Heptagon','Octagon','Nonagon','Decagon']

while n<11: # limiting to a decagon

trtl.clear()

for i in range(n): # for loop to minimize the same lines of codes being written

trtl.pencolor('red')

trtl.forward(60) #top line

trtl.right(360/n)

n=n+1

trtl.penup()

trtl.home()

trtl.pendown()

trtl.penup()

trtl.home() #moving the turtle to make the animation more centric

trtl.pendown()

trtl.ht()

time.sleep(1)

trtl.st()

.

23. Write a program illustrating the object oriented features supported by Python.

Aim:

Program:

**Creating an Object and Class in python:**

class employee():

def \_\_init\_\_(self,name,age,id,salary): #creating a function

self.name = name # self is an instance of a class

self.age = age

self.salary = salary

self.id = id

emp1 = employee("harshit",22,1000,1234) #creating objects

emp2 = employee("arjun",23,2000,2234)

print("employee details are: ",emp1.\_\_dict\_\_) #Prints dictionary

print("employee id is: ",emp2.id)

**Creating a Multilevel Inheritance in python:**

class employee(): #Super class

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

self.name = name

self.age = age

self.salary = salary

class childemployee1(employee) :#First child class

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

self.name = name

self.age = age

self.salary = salary

class childemployee2(childemployee1): #Second child class

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

self.name = name

self.age = age

self.salary = salary

emp1 = employee('harshit',22,1000)

emp2 = childemployee1('arjun',23,2000)

emp3 = childemployee2('sagar',27,5000)

print("employee age is: ",emp1.age)

print("employee age is: ",emp2.age)

print("employee salary is: ",emp2.salary)

25. Design a Python script to determine the difference in date for given two dates in YYYY:MM:DD format(0 <= YYYY <= 9999, 1 <= MM <= 12, 1 <= DD <= 31) following the leap year rules.

Aim:

Program:

# importing datetime module

from datetime import \*

# Enter birth dates and store

# into date class objects

y1, m1, d1 = [int(x) for x in input("Enter first person's date(YYYY/MM/DD): ").split('/')]

Date1 = date(y1, m1, d1)

# Input for second date

y2, m2, d2 = [int(x) for x in input("Enter second person's date(YYYY/MM/DD): ").split('/')]

Date2 = date(y2, m2, d2)

# Check the dates

if Date1 == Date2:

print("Both persons are of equal age")

else:

diff=Date1-Date2

print(“age differenc is: ”,diff)