VISHWA BHARATI PUBLIC SCHOOL,NOIDA

2024-2025

This is to certify that Vaibhav Sharma

Roll Number: 35 of Class 12.C has successfully completed Kronos.AI- The Physics Guide according to the guidelines laid down by CBSE.

Teacher Incharge

Principal

ACKNOWLEDGEMENT

We deeply acknowledge the success of this project to my Computer Science teacher Ms. Sofia Goel, who provided us with this golden and creative opportunity to conduct such an amazing project.

It enabled us all to get first-hand research experience in a friendly peer environment to inculcate crucial skills which would be quintessential in the long run.

Lastly, we would like to extend our thanks to the team who worked on this project and successfully completed it in this limited time frame.

CONTENTS

Signature

Introduction

In this world of rapid technological change and automation, a student may feel the need for visualizations and a problem-solving guide for his or her in his academics, especially for a core centric concept like physics. So, wouldn't it be easy to scroll many videos or sites over an automation which can help people with the confusing variables and basic questions?

In line with this vision, we have developed a Physics Chatbot which can be very helpful in teaching elementary physics. Physics is considered as a visually challenging and memory application driven field. So, we have attempted to create a data-oriented bot which can generate responses automatically with respect to specific keywords. So let us dive into it.

OBJECTIVE

The objective of this project is to create an intelligent Physics Chatbot that serves as an interactive educational assistant for students learning elementary physics.

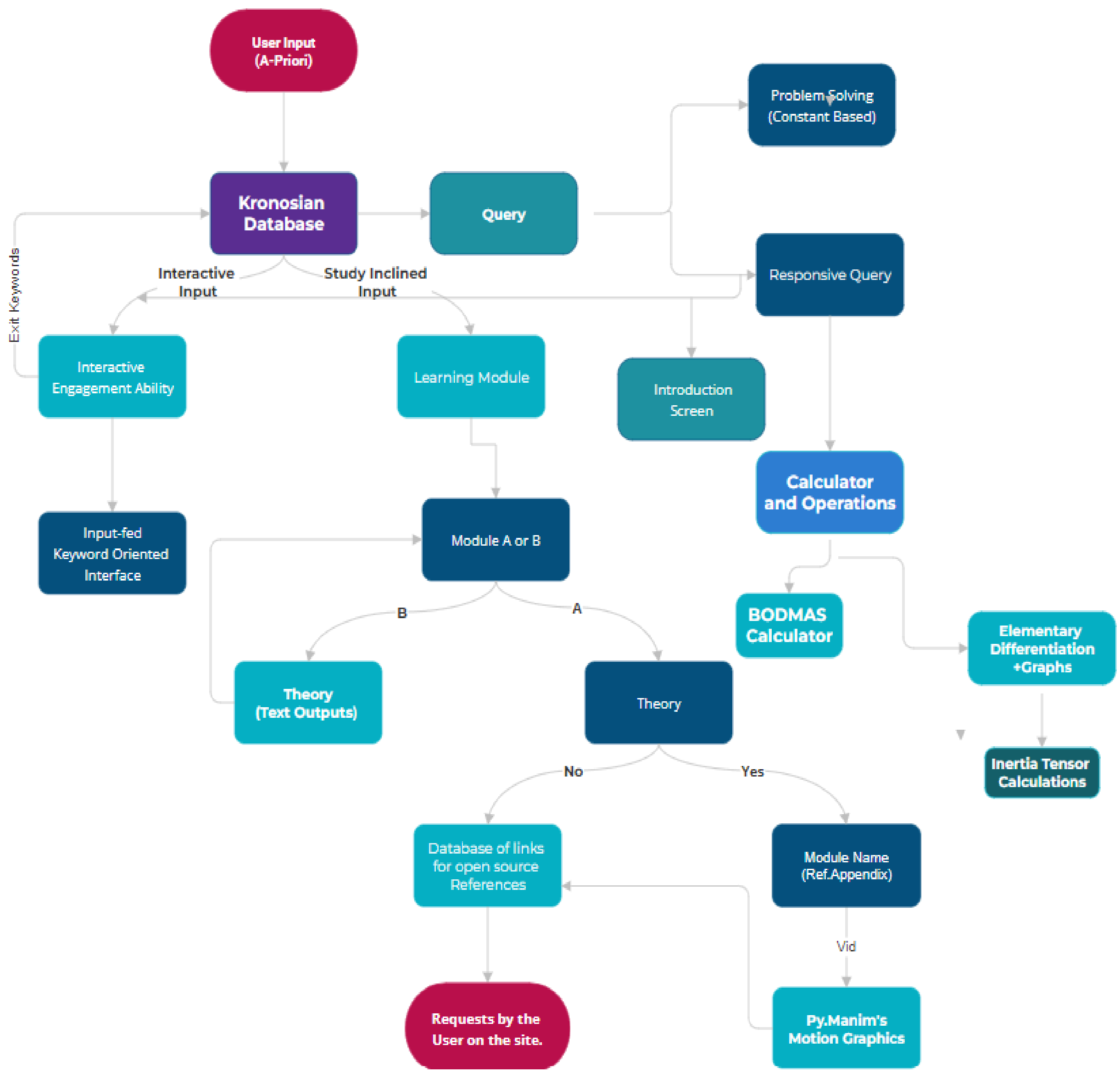
The chatbot will deliver clear and concise explanations of fundamental physics concepts, offering relevant formulas, variables, and detailed information tailored to each topic. To enhance the learning experience, the chatbot will provide visual aids, such as elementary graphics and diagrams, to illustrate complex ideas in a simplified manner.

Additionally, the chatbot will integrate a powerful problem-solving tool, capable of solving physics problems step-by-step, along with a built-in calculator for calculations and a unit converter for accurate and seamless conversions.

By combining interactive dialogue, visual learning, and practical tools, the chatbot aims to provide a comprehensive and user-friendly platform that makes learning physics more engaging, intuitive, and accessible for students.

OVERVIEW

The Physics Chatbot (Codenamed- Kronos), is an input based Physics Guide which is designed to interactively guide the user through a multitude of Physics concepts.



The user’s latest and first query or inputs are considered as the ‘A-Priori’ by the chatbot and the response is keyword oriented. The subsequent queries are taken as an ‘A Posteriori’.

The responses are guided as pathways to an intricate pathway of queries and are subdivided as follows-

1) Interactive Query

Kronos is designed to be an interactive platform and responds to personal messages via key-pairing with an inbuilt miniature database to generate random outputs and play along with the user afresh for each use.

The ‘Bye’ response and associated keywords help in discontinuing the programme at the user’s wish. The user can thoroughly navigate through the various commands via Kronos’s Introductory Dialogue box, which aims at the user’s self-sufficiency to increase engagement.

2) Responsive Query

These queries are far more subject specific and reconvened to specific uses namely-

- Study Based Query

With the use of specific keywords, Kronos can be used to navigate through it’s study modules, which provide the following resources-

➔ Text Files for Theoretical Data
➔ SQL Database of Open Source Web Resources
➔ Illustrative Motion Graphics (made by Py.Manim)
➔ Generally used Constants

- Theoretical Query
- Graphical Query

Modules like Scipy,Sympy, Numpy and Mathplotlib are used for the proper and accurate representation of user defined functions in the Taylor-Maclaurin Series Expansion Pack Module. (App.A)

3) Operational Query

This involves a handy BODMAS Calculator as well as a Derivative calculator. (Refer to App.B)

CODE

```
# Kronos.AI Chatbot

# Importing the nesseacry libraries and modules
import pyspark
import seaborn
import scipy
import cv2
import manim
import astropy
import mysql

import io
import random
import string
import warnings
from turtle import *
from PIL import Image
from tabulate import tabulate
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import nltk
from nltk.stem import WordNetLemmatizer

warnings.filterwarnings('ignore')

nltk.download('popular', quiet=True)
nltk.download('punkt')
nltk.download('wordnet')

# Primary Corpus of the Chatbot

with open('chatbot.txt', 'r', encoding='utf8', errors='ignore') as fin:
    raw = fin.read().lower()

# Raw Tokenization

sent_tokens = nltk.sent_tokenize(raw)
word_tokens = nltk.word_tokenize(raw)

lemmer = WordNetLemmatizer()


def LemTokens(tokens):
    return [lemmer.lemmatize(token) for token in tokens]
```

```python
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))


# Keyword Pairing Algorithm


GREETING_INPUTS = ("hello", "hi", "greetings", "sup", "what's up", "hey", "hola")
GREETING_RESPONSES = ["Hello Gentleman", "Hi", "Greetings mate!", "Cheers!", "Good Day, how may I help you out?", "What brings you here today?", "Just say the word.."]

def greeting(sentence):
    """If the user's input is a greeting, return a greeting response"""
    for word in sentence.split():
        if word.lower() in GREETING_INPUTS:
            return random.choice(GREETING_RESPONSES)


# Answer Genertaion

def response(user_response):
    robo_response = ''
    sent_tokens.append(user_response)
    TfidfVec = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english')
    tfidf = TfidfVec.fit_transform(sent_tokens)
    vals = cosine_similarity(tfidf[-1], tfidf)
    idx = vals.argsort()[0][-2]
    flat = vals.flatten()
    flat.sort()
    req_tfidf = flat[-2]
    if req_tfidf == 0:
        robo_response = robo_response + "Aplogies! I can't understand your input. Please try capitalising the first letter of your input after every space.(Example- velocity to Velocity)
        return robo_response
    else:
        robo_response = robo_response + sent_tokens[idx]
        return robo_response


#Appropriate Variable providing Dictionary
```

```
#Appropriate Variable providing Dictionary

dict_Variables = {
    "Distance": 'x',
    "Displacement": 's',
    "Time": 't',
    "Mass": 'm',
    "Energy": 'E',
    "Kinetic Energy": 'Ek',
    "Potential Energy": 'Ep',
    "Spring Constant": 'K',
    "Boltzmann Constant": 'k',
    "Universal Gas Constant": 'R',
    "Wien Constant": 'σ',
    "Planck Constant": 'h',
    "Torque": 'τ',
    "Coefficient of Viscosity": 'η',
    "Modulus of Rigidity": 'η',
    "Universal Gravitation Constant": 'G',
    "Young Modulus": 'Y',
    "Stress": 'σ',
    "Strain": 'ε',
    "Stefan Constant": 'θ',
    "Dielectric Constant": 'ε',
    "Pressure": 'P',
    "Volume": 'V',
    "Area": 'A',
    "Momentum": 'p',
    "Angular Momentum": 'L',
    "Acceleration": 'a',
    "Velocity": 'v',
    "Angular Velocity": 'ω',
    "Angular Acceleration": 'α',
    "Density": 'ρ',
    "Angular Wave Number": 'k',
    "Coefficient of Friction": 'μ',
    "Refractive Index": 'μ',
    "Wave Number": 'ν',
    "Frequency": 'ν',
    "Time Period": 'T',
    "Wavelength": 'λ',
    "Decay Constant": 'λ',
    "Permittivity of Free Space": 'μ',
    "Rate Constant": 'k',
    "Electrical Field": 'E',
```

```
resized_image.save("resized_background.png")


screen = Screen()
screen.bgpic("resized_background.png")

hideturtle()
speed("fast")# Kronos.AI Chatbot
```

```python
# Run the program for Greek Variable Table

    elif user_response == 'greek_variable_list':
        Greek_Variables = [
            ['Variables', 'Pronunciation'],
            ['α', 'Alpha'],
            ['β', 'Beta'],
            ['γ', 'Gamma'],
            ['δ', 'Delta'],
            ['Δ', 'Delta'],
            ['ε', 'Epsilon'],
            ['η', 'Eta'],
            ['φ', 'Phi'],
            ['ι', 'Iota'],
            ['κ', 'Kappa'],
            ['λ', 'Lambda'],
            ['μ', 'Mu'],
            ['ν', 'Nu'],
            ['ξ', 'Upsilon'],
            ['ρ', 'Rho'],
            ['π', 'Pi'],
            ['σ/ς', 'Sigma'],
            ['ψ', 'Psi'],
            ['ω', 'Omega'],
            ['ζ', 'Zeta'],
            ['θ', 'Theta'],
            ['τ', 'Tau']
        ]

        print(tabulate(Greek_Variables, headers="firstrow", tablefmt='fancy_grid'))
```

```python
def inertia_tensors(x, weights=None):


    #Preludes- (Egs)

    x, weights = _process_args(x, weights)
    n1, n2, ndim = np.shape(x)

    I = np.einsum('...ij,...ik->...jk', x, x*weights)
    m = np.sum(weights, axis=1)
    return I/(np.ones((n1,ndim,ndim))*m[:,np.newaxis])


def reduced_inertia_tensors(x, weights=None):

    x, weights = _process_args(x, weights)
    n1, n2, ndim = np.shape(x)

    r_squared = np.sum(x**2, -1)

    # ignore points at r=0
    mask = (r_squared==0.0)
    weights[mask] = 0.0
    r_squared[mask] = 1.0

    I = np.einsum('...ij,ik->...jk', x/(r_squared[:,:,np.newaxis]), x*weights)
    m = np.sum(weights, axis=1)
    return I/(np.ones((n1,ndim,ndim))*m[:,np.newaxis])


def iterative_inertia_tensors_3D(x, weights=None, rtol=0.01, niter_max=5):

    x, weights = _process_args(x, weights)
    n1, n2, ndim = np.shape(x)

    rot_func = rotation_matrices_from_basis_3d
```

```
# intial ellipsoidal volume
ellipsoid_volume_0 = (4.0/3.0)*np.pi*A*B*C

# intial axis ratios
b_to_a_0, c_to_a_0 = B/A, C/A
Av_0 = Av

niter = 1  # iteratively calculate I
exit=False
while (niter < niter_max) & (exit==False):

    # calculate rotation matrix between eigen basis and axis-aligned basis
    rot = rot_func(Av, Bv, Cv)
    inv_rot = np.linalg.inv(rot)

    # rotate distribution to align with axis
    xx = rotate_vector_collection(inv_rot, x)

    # calculate ellipsoidal radial distances
    axis_ratios = np.vstack((A,B,C)).T
    norm = np.repeat(axis_ratios[:,np.newaxis,:], n2, axis=1)
    r_squared = np.sum((xx/norm)**2, -1)

    # ignore points at r=0
    mask = (r_squared==0.0)
    weights[mask] = 0.0
    r_squared[mask] = 1.0

    # calculate eigen tensors
    I = np.einsum('...ij,...ik->...jk', xx/(r_squared[:,:,np.newaxis]), xx*weights)
    m = np.sum(weights, axis=1)
    I = I/(np.ones((n1,ndim,ndim))*m[:,np.newaxis])

    A, B, C, Av, Bv, Cv = _principal_axes_3D(I)

    # rotate back into original frame
    Av = rotate_vector_collection(rot, Av)
    Bv = rotate_vector_collection(rot, Bv)
    Cv = rotate_vector_collection(rot, Cv)

    # re-scale axes to maintain constant volume
    ellipsoid_volume = (4.0/3.0)*np.pi*A*B*C
    f = (1.0*ellipsoid_volume/ellipsoid_volume_0)
    A = A*f**(-1.0/3.0)
    B = B*f**(-1.0/3.0)
    C = C*f**(-1.0/3.0)
```

```
        # angle between primary eigenvectors
        theta = np.degrees(angles_between_list_of_vectors(Av, Av_0))

        # update parameters
        b_to_a_0 = b_to_a
        c_to_a_0 = c_to_a
        Av_0 = Av
        niter += 1

    # re-construct inertia tensor
    m = np.tile(np.identity(3), (n1,1,1))
    m[:,0,0] = A**2
    m[:,1,1] = B**2
    m[:,2,2] = C**2

    s = np.zeros((n1,3,3))
    s[:,:,0] = Av
    s[:,:,1] = Bv
    s[:,:,2] = Cv

    I = np.matmul(np.matmul(s,m),s.transpose(0,2,1))

    # check reconstruction
    evals, evecs = np.linalg.eigh(I)
    assert np.allclose(np.sqrt(evals[:,0]),C)
    assert np.allclose(np.sqrt(evals[:,1]),B)
    assert np.allclose(np.sqrt(evals[:,2]),A)

    return I

f user response== 'Derivatives' or 'differentiation':
    x, y = symbols('x y')
xpr = x**2 + 2 * y + y**3
rint("Expression : {}".format(expr))
xpr_diff = Derivative(expr, x)

rint("Derivative of expression with respect to x : {}".format(expr_diff))
rint("Value of the derivative : {}".format(expr_diff.doit()))

f user_response == 'Graph'

ef Taylor_Series(F1,F2,F3):
    if user_input == 'Taylor Series' or 'Maclaurin Series':
```

```
print("Derivative of expression with respect to x : {}".format(expr_diff))
print("Value of the derivative : {}".format(expr_diff.doit()))

if user_response == 'Graph'

def Taylor_Series(F1,F2,F3):
    if user_input == 'Taylor Series' or 'Maclaurin Series':
        print('Welcome to the Taylor- Maclaurin Theorem')
        f1= f1.open('Taylor-Maclaurin Expansion Pack.txt')


import matplotlib.pyplot as mplt
import numpy as np

x = np.linspace(-20, 20, 90)
fig = mplt.figure(figsize = (14, 8))
ax= int(input('How many functions do you wish to input:'))

for i in range(0 ,ax +1):
    y = np.sin(x)+1
    mplt.plot(x, y, 'b', label ='sin(x)+1')

    y2 = (x**3 / 3)-(x**3)
    mplt.plot(x, y2, 'r-.', label ='Degree 3')

    y4 = 1 - x**2 / 2 + x**4 / 24

    mplt.plot(x, y4, 'g:', label ='Degree 4')





# Add features to our figure

mplt.legend()
mplt.grid(True, linestyle ='-')
mplt.xlim([-6, 6])
mplt.ylim([-4, 4])

mplt.title('Taylor Polynomials of arccosh(x) at x = 0')
mplt.xlabel('x-axis')
mplt.ylabel('y-axis')
```

```
# Show plot
mplt.show()
II2= input('Which module do you wish to choose: Theory for A and references for B')
 if I2 =='A':
     I23= input('Which module:')
     if I23 =='Bermoulii':
         path= r'C:\Users\Shalini\OneDrive\Desktop\CS Project 2024\CS Project\Final Subprogrammes\Bernoullis_Equation - Made with Clipchamp.mp4'
cap = cv2.VideoCapture(path)

if (cap.isOpened()== False):
    print("Error opening video file")

print(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
print(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

ret = cap.set(cv2.CAP_PROP_FRAME_WIDTH,320)
ret = cap.set(cv2.CAP_PROP_FRAME_HEIGHT,240)



# Read until video is completed

while(cap.isOpened()):

# Capture frame-by-frame
    ret, frame = cap.read()
    if ret == True:
    # Display the resulting frame
        cv2.imshow('Frame', frame)

        def make_1080p():
            cap.set(3, 1920)
            cap.set(4, 800)



        def make_720p():
            cap.set(3, 800)
            cap.set(4, 500)

        def make_480p():
            cap.set(3, 640)
            cap.set(4, 480)

        def change_res(width, height):
            cap.set(3, width)
```

```
# Capture frame-by-frame
    ret, frame = cap.read()
    if ret == True:
    # Display the resulting frame
        cv2.imshow('Frame', frame)

        def make_1080p():
            cap.set(3, 1920)
            cap.set(4, 800)



        def make_720p():
            cap.set(3, 800)
            cap.set(4, 500)

        def make_480p():
            cap.set(3, 640)
            cap.set(4, 480)

        def change_res(width, height):
            cap.set(3, width)
            cap.set(4, height)

        make_720p()

        change_res(800, 500)

    # Press Q on keyboard to exit
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break

    else:
        break
cap.release()
cv2.destroyAllWindows()
 else:
     db=mysql.connect(host ='localhost', user='root', password= 'Vaibhav_@Sql', Database='Physics sources')
     cursor=db.cursor()
     a2=cursor.execute(desc.database(Physics_Refrences)
    print(a2)
        db.close()
```

```
        saturn = Planet("Saturn", 230, 'burlywood3')
        uranus = Planet("Uranus", 250, 'Cyan3')
        neptune = Planet("Neptune", 280, 'DarkTurquoise')

        # Planet list
        planet_List = [mercury, venus, earth, mars, jupiter, saturn, uranus, neptune]

        while True:
            screen.update()
            for i in planet_List:
                i.move()

            # Increasing the angles by 0.0x radians

            mercury.angle += 0.05
            venus.angle += 0.03
            earth.angle += 0.01
            mars.angle += 0.007
            jupiter.angle += 0.02
            saturn.angle += 0.018
            uranus.angle += 0.016
            neptune.angle += 0.005
            break
        break
    break
if user_response == 'unitconverter':
```

```
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic meters' and result_to == 'Liters':
        calculate = number1*1000
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic meters' and result_to == 'Gallons':
        calculate = number1*264.172
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic meters' and result_to == 'Cubic centimeters':
        calculate = number1*1000000
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic foot' and result_to == 'Cubic meters':
        calculate = number1*0.02831
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic foot' and result_to == 'Cubic foot':
        calculate = number1
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic foot' and result_to == 'Liters':
        calculate = number1*28.31679
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic foot' and result_to == 'Gallons':
        calculate = number1*7.4805
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic foot' and result_to == 'Cubic centimeters':
        calculate = number1*28316.8
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Liters' and result_to == 'Cubic meters':
```

```
        result.configure(text = (calculate,result_to))

    elif result_from == 'Liters' and result_to == 'Cubic foot':
        calculate = number1*0.0353146
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Liters' and result_to == 'Liters':
        calculate = number1
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Liters' and result_to == 'Gallons':
        calculate = number1*0.26417
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Liters' and result_to == 'Cubic centimeters':
        calculate = number1*1000
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Gallons' and result_to == 'Cubic meters':
        calculate = number1*0.003785
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Gallons' and result_to == 'Cubic foot':
        calculate = number1*0.13368
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Gallons' and result_to == 'Liters':
        calculate = number1*3.7854
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Gallons' and result_to == 'Gallons':
        calculate = number1
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Gallons' and result_to == 'Cubic centimeters':
        calculate = number1*3786.41
        result.cget('text')
        result.configure(text = (calculate,result_to))
```

```python
    number_from = StringVar()

    def fromfunc(event):
        global result_from
        result_from = event.widget.get()

    def tofunc(event):
        global result_to
        result_to = event.widget.get()


# The answer function
def answer(n1):
    num1 = n1.get()
    try:
        number1 = int(num1)
    except:
        messagebox.showerror('Error','Term not recognised')

    # Length
    if result_from == 'Cubic meters' and result_to == 'Cubic meters':
        calculate = number1
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic meters' and result_to == 'Cubic foot':
        calculate = number1*35.3147
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic meters' and result_to == 'Liters':
        calculate = number1*1000
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic meters' and result_to == 'Gallons':
        calculate = number1*264.172
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic meters' and result_to == 'Cubic centimeters':
        calculate = number1*1000000
```

```
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic meters' and result_to == 'Cubic centimete
        calculate = number1*1000000
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic foot' and result_to == 'Cubic meters':
        calculate = number1*0.02831
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic foot' and result_to == 'Cubic foot':
        calculate = number1
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic foot' and result_to == 'Liters':
        calculate = number1*28.31679
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic foot' and result_to == 'Gallons':
        calculate = number1*7.4805
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic foot' and result_to == 'Cubic centimeters
        calculate = number1*28316.8
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Liters' and result_to == 'Cubic meters':
        calculate = number1*0.000999
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Liters' and result_to == 'Cubic foot':
        calculate = number1*0.0353146
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Liters' and result_to == 'Liters':
        calculate = number1
        result.cget('text')
```

```
elif result_from == 'Liters' and result_to == 'Gallons':
    calculate = number1*0.26417
    result.cget('text')
    result.configure(text = (calculate,result_to))

elif result_from == 'Liters' and result_to == 'Cubic centimeters':
    calculate = number1*1000
    result.cget('text')
    result.configure(text = (calculate,result_to))

elif result_from == 'Gallons' and result_to == 'Cubic meters':
    calculate = number1*0.003785
    result.cget('text')
    result.configure(text = (calculate,result_to))

elif result_from == 'Gallons' and result_to == 'Cubic foot':
    calculate = number1*0.13368
    result.cget('text')
    result.configure(text = (calculate,result_to))

elif result_from == 'Gallons' and result_to == 'Liters':
    calculate = number1*3.7854
    result.cget('text')
    result.configure(text = (calculate,result_to))

elif result_from == 'Gallons' and result_to == 'Gallons':
    calculate = number1
    result.cget('text')
    result.configure(text = (calculate,result_to))

elif result_from == 'Gallons' and result_to == 'Cubic centimeters':
    calculate = number1*3786.41
    result.cget('text')
    result.configure(text = (calculate,result_to))

elif result_from == 'Cubic centimeters' and result_to == 'Cubic meters':
    calculate = number1*9.99999
    result.cget('text')
    result.configure(text = (calculate,result_to))
```

```
    elif result_from == 'Cubic centimeters' and result_to == 'Cubic foot':
        calculate = number1*3.53146
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic centimeters' and result_to == 'Liters':
        calculate = number1*0.000999
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic centimeters' and result_to == 'Cubic meters':
        calculate = number1*9.9999
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic centimeters' and result_to == 'Liters':
        calculate = number1*0.00099999
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic centimeters' and result_to == 'Gallons':
        calculate = number1*0.00026417
        result.cget('text')
        result.configure(text = (calculate,result_to))

    elif result_from == 'Cubic centimeters' and result_to == 'Cubic centimeters':
        calculate = number1
        result.cget('text')
        result.configure(text = (calculate,result_to))

def selected(event):
    unit = event.widget.get()
    if unit == 'Volume':
        fromdd['values'] = ('Cubic meters',
                            'Cubic foot',
                            'Liters',
                            'Gallons',
                            'Cubic centimeters')

        todd['values'] = ('Cubic meters',
                          'Cubic foot',
                          'Liters',
                          'Gallons',
                          'Cubic centimeters')
```

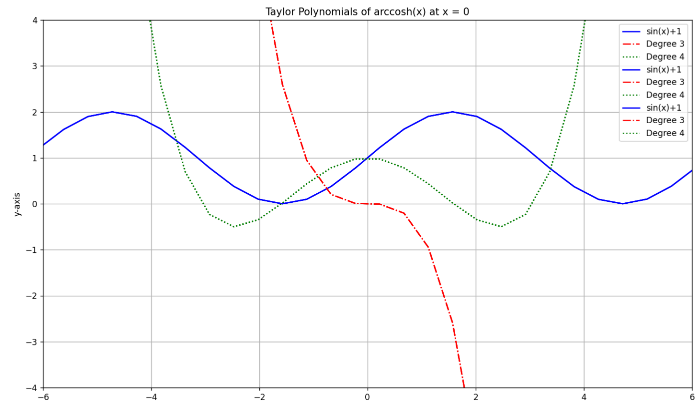
```python
def selected(event):
    unit = event.widget.get()
    if unit == 'Volume':
        fromdd['values'] = ('Cubic meters',
                            'Cubic foot',
                            'Liters',
                            'Gallons',
                            'Cubic centimeters')

        todd['values'] = ('Cubic meters',
                          'Cubic foot',
                          'Liters',
                          'Gallons',
                          'Cubic centimeters')

    elif unit == 'Length':
        fromdd['values'] = ('Millimeters',
                            'Centimeters',
                            'Decimeters',
                            'Meters',
                            'Kilometers')

        todd['values'] = ('Millimeters',
                          'Centimeters',
                          'Decimeters',
                          'Meters',
                          'Kilometers')

    elif unit == 'Mass':
        fromdd['values'] = ('Milligrams',
                            'Centigrams',
                            'Grams',
                            'Decigrams',
                            'Kilograms')

        todd['values'] = ('Milligrams',
                          'Centigrams',
                          'Grams',
                          'Decigrams',
                          'Kilograms')
```

```python
main = tk.Label(window,text = 'Unit Converter',bg = 'peach puff2',fg = 'blue')
main['font'] = font1
main.place(relx = '0.48',rely = '0.1',anchor = 'center')

# Creating the unit label
unit = tk.Label(window,text = 'Unit -:',bg = 'peach puff2')
unit['font'] = font2
unit.place(relx = '0.25',rely = '0.28')

n = StringVar()
unitdd = ttk.Combobox(window,width = '35',textvariable = n)

# Values
unitdd['values'] = ('Volume',
                    'Length',
                    'Mass')

unitdd.place(relx = '0.57',rely = '0.3',anchor = 'center')
unitdd.current()
unitdd.bind('<<ComboboxSelected>>',selected)

label_from = tk.Label(window,text = 'From -:',bg = 'peach puff2')
label_from['font'] = font2
label_from.place(relx = '0.238',rely = '0.37')

f = StringVar()
fromdd = ttk.Combobox(window,width = '35',textvariable = f)

fromdd.place(relx = '0.57',rely = '0.39',anchor = 'center')
fromdd.current()
fromdd.bind('<<ComboboxSelected>>',fromfunc)
num_from = tk.Entry(window,width = 10,textvariable = number_from)
num_from.place(relx = '0.82',rely = '0.37')
answer = partial(answer,num_from)
to = tk.Label(window,text = 'To -:',bg = 'peach puff2')
to['font'] = font2
to.place(relx = '0.268',rely = '0.45')
t = StringVar()
todd = ttk.Combobox(window,width = 35,textvariable = t)

todd.place(relx = '0.57',rely = '0.47',anchor = 'center')
todd.current()
todd.bind('<<ComboboxSelected>>',tofunc)
result = tk.Label(window,text = '',bg= 'white',width = 20)
result['font'] = font3
```

```
f = StringVar()
fromdd = ttk.Combobox(window,width = '35',textvariable = f)

fromdd.place(relx = '0.57',rely = '0.39',anchor = 'center')
fromdd.current()
fromdd.bind('<<ComboboxSelected>>',fromfunc)
num_from = tk.Entry(window,width = 10,textvariable = number_from)
num_from.place(relx = '0.82',rely = '0.37')
answer = partial(answer,num_from)
to = tk.Label(window,text = 'To -:',bg = 'peach puff2')
to['font'] = font2
to.place(relx = '0.268',rely = '0.45')
t = StringVar()
todd = ttk.Combobox(window,width = 35,textvariable = t)

todd.place(relx = '0.57',rely = '0.47',anchor = 'center')
todd.current()
todd.bind('<<ComboboxSelected>>',tofunc)
result = tk.Label(window,text = '',bg= 'white',width = 20)
result['font'] = font3
result.place(relx = '0.21',rely = '0.6')
get_answer = tk.Button(window,text = 'Get Answer',bg = 'cyan2',command = answer)
get_answer['font'] = font2
get_answer.place(relx = '0.46',rely = '0.7')
art = tk.Label(window,text = 'The Art Of Programming',bg= 'peach puff2',fg = 'blue')
art['font'] = font3
art.place(relx = '0.21',rely = '0.9')
window.mainloop()
break
    elif user_response != 'bye':
        if user_response == 'thanks' or user_response == 'thank you':
            flag = False
            print("KRONOS: Told you! I would be handy..")
        else:
            if greeting(user_response) is not None:
                print("KRONOS:" + greeting(user_response))
            else:
                print("KRONOS:", end="")
                print(response(user_response))
                sent_tokens.remove(user_response)
    else:
        flag = False
        print("Thanks, Programme Deactivated")
```

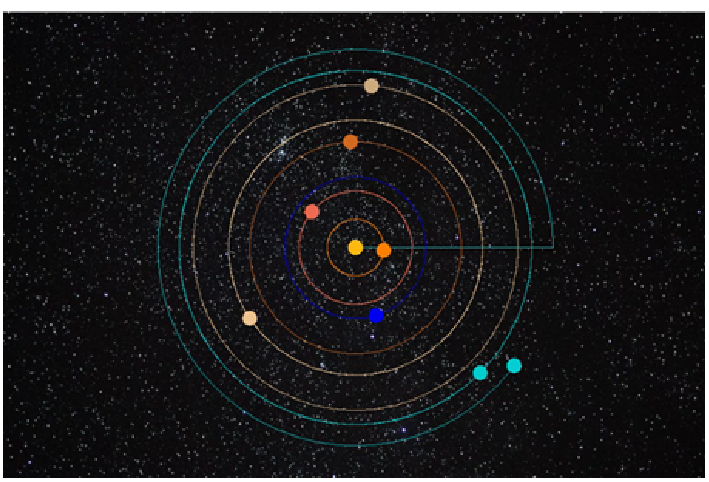
OUTPUT

Kronos's Introduction Screen



Graphical Output

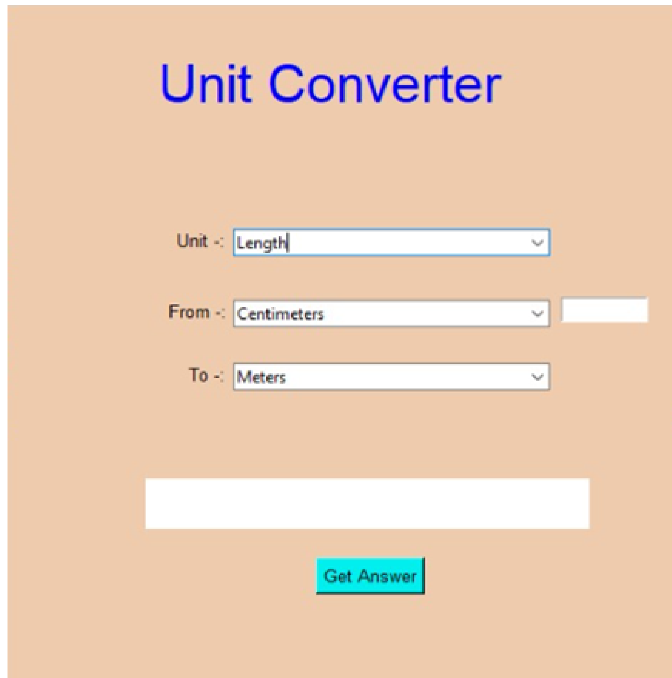
Solar System Model



Variables	Pronunciation
α	Alpha
β	Beta
γ	Gamma
δ	Delta
Δ	Delta
ε	Epsilon
η	Eta
φ	Phi
ι	Iota
κ	Kappa
λ	Lambda
μ	Mu
ν	Nu

ROBO: My name is Robo. I will answer any queries you may have. If you want to end this conversation, type Bye!
yinyang
unit converter
Time
The variable for 'Time' is: t
Displacement
The variable for 'Displacement' is: s
Velocity
The variable for 'Velocity' is: v
Wien Constant
The variable for 'Wien Constant' is: b
Stress
The variable for 'Stress' is: σ
Density
The variable for 'Density' is: ρ
Momentum
The variable for 'Momentum' is: p

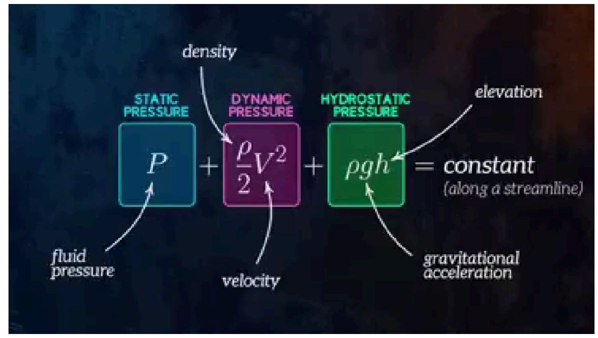
```
Derivatives.py" "
Expression : x**2 + y**3 + 2*y
Derivative of expression with respect to x : Derivative(x**2 + y**3 + 2*y, x)
Value of the derivative : 2*x
```



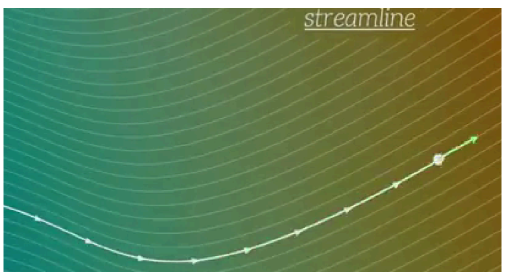

```
Taylor- Maclaurin Series Expansion Pack

History of Taylor and Maclaurin Series

Given By- Brook Taylor

The concept of the Taylor series is attributed to the English mathematician Brook Taylor. He introduced it in his work "Methodus Incrementorum Directa et Inversa," published in 1715.
Taylor's series is a way to represent functions as infinite sums of terms calculated from the values of their derivatives at a single point. This idea was revolutionary because it allowed
mathematicians to approximate complex functions using simpler polynomial expressions. Taylor's work laid the foundation for many advances in calculus and analysis.

Colin Maclaurin
Colin Maclaurin, a Scottish mathematician, extended Taylor's work. Maclaurin's series is a special case of the Taylor series, centered at zero. Maclaurin's contributions to mathematics
were substantial, and his work on series was published posthumously in 1742. The Maclaurin series allows for simpler expansions and is often used when functions are centered around the
origin.

Further Developments
Over the years, the concepts of Taylor and Maclaurin series were refined and expanded by many mathematicians. Joseph-Louis Lagrange provided a rigorous foundation for the use of series in
calculus. Augustin-Louis Cauchy, Karl Weierstrass, and others further developed the theoretical underpinnings and applications of series in mathematical analysis.

Mathematical Formulation
Taylor Series
The Taylor series of a function

(

)
 around a point

 is given by: $$ f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (x - a)^n $$ where

()
 denotes the

-th derivative of

 evaluated at

.

This series can be understood as constructing a polynomial that approximates the function
```

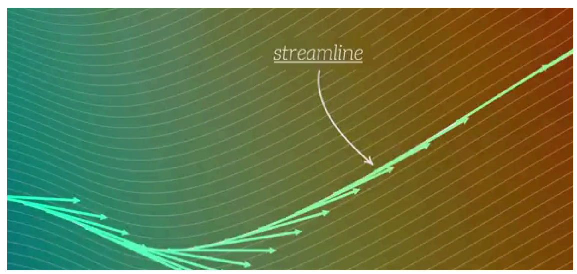
Bernoulli's Theorem Package-

Equational Graphics



Manim Graphics-



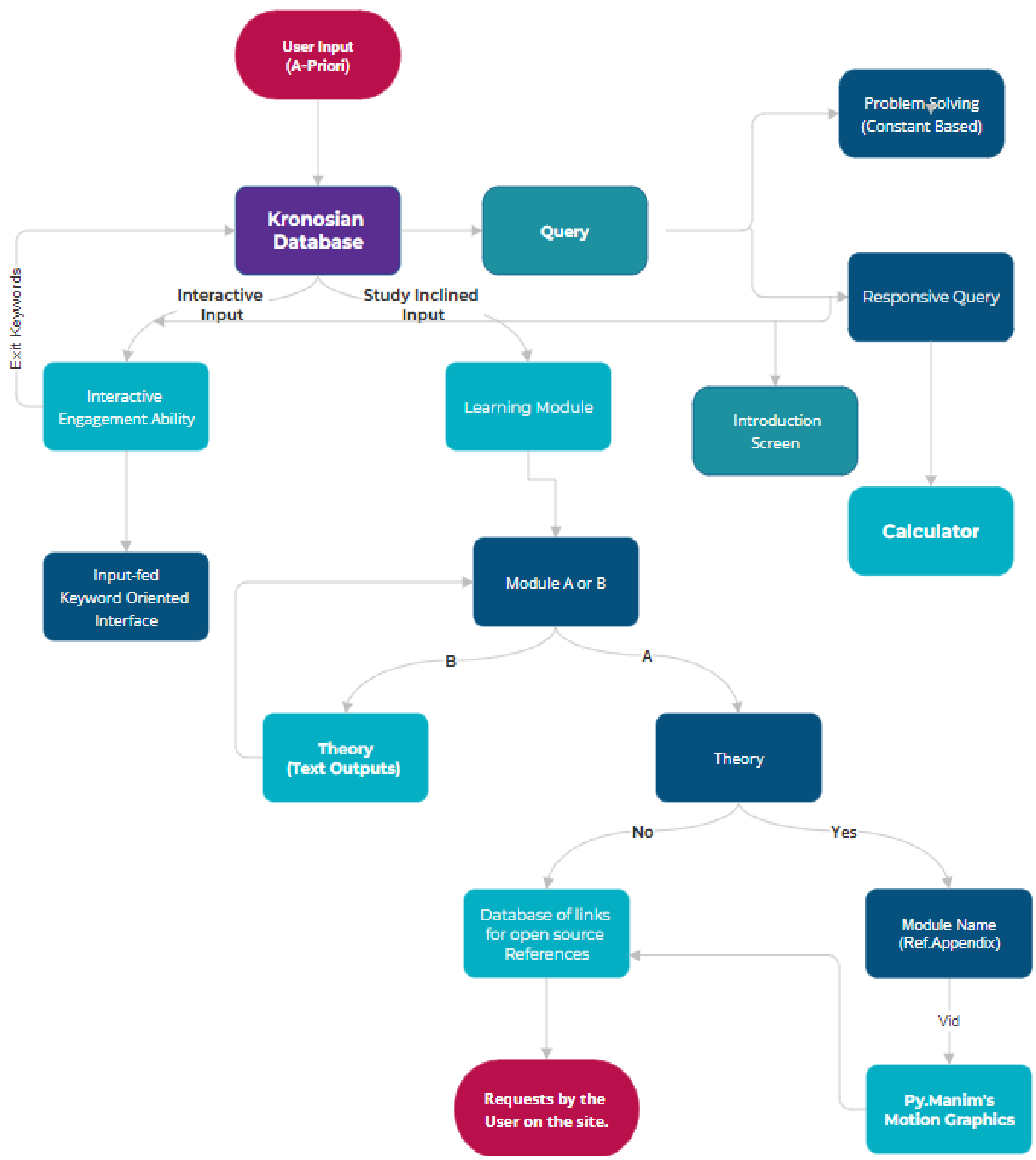


Before

After

APPENDICES

- Simplified Flowchart-

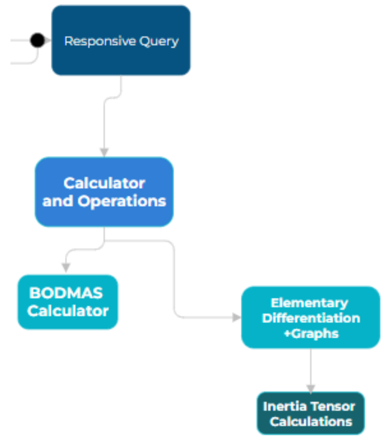


Appendix A-

1) Response Query Structure-

It is divided into 3 structures as follows-

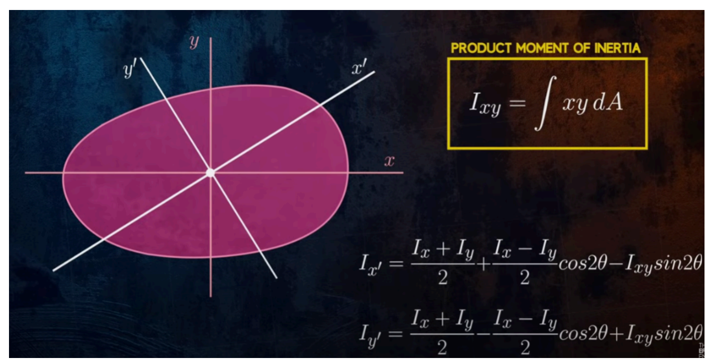
- Introductory Screen
- Bodmas Calculator
- Differentiation (C.N.-Syctonius)
- Inertia Tensor



1) Inertia Tensor-

It involves the use of dividing the dimensions equally into arrays of equal areas and averaging the areas out. The Integration involved is complex and thus is limited to only regular shapes.

The Integration done is the average summation done of the 3 tensoral arrays calculated with the internal data of the Sympy module and the given data.



'The (symmetric) matrix representing the inertia tensor of a collection of masses, relative to their centre of mass is-

$$\mathbf{I} = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{pmatrix},$$

where

$$I_{xx} = \sum_i m_i(y_i^2 + z_i^2), \qquad I_{yy} = \sum_i m_i(x_i^2 + z_i^2), \qquad I_{zz} = \sum_i m_i(x_i^2 + y_i^2),$$

$$I_{xy} = -\sum_i m_i x_i y_i, \qquad I_{yz} = -\sum_i m_i y_i z_i, \qquad I_{xz} = -\sum_i m_i x_i z_i.$$

'

-scipython.com/book/chapter-6-numpy/problems/p65/the-moment-of-inertia-tensor/

2) Differentiation-

Code Name- Syctonius

It is a simplex involving Sympy and Scipy to calculate the differentiation of given functions via the existing module formulas and iteration of array logic.

The Graphing of these derivative functions is a part of the broader Taylor-Maclaurin Expansion Pack' , as discussed in Appendix-'B'.

2) Appendix B-

The developed modules uptill now are as follows-

1) Taylor-Maclaurin Expansion Pack-

It attempts to explain the approximate conversion of functions developed by Sir Brook Taylor. The module provides a theory as well as the theory in a text file. The differentiation module and graphing gives the application of any given function.

2) Inertia Tensor

(Discussed in Appendix A)

3) Bernoulli's Equation-

The graphics are created via py.manim, which is a user made module for creating motion graphics. Due to operational limitations, the graphics are converted into a video and integrated by the CV2 Module.

4) Solar System Module-

This module defines the heliocentric view of the solar system via the 'Turtle Module' and uses values close to the original approximation values.

BIBLIOGRAPHY

1) Books-

- Class XI and XII NCERT Books (Computer Sc.)
- CS with Python (by Preeti Arora)
- Learn Python the Hard Way! (by Zed Shaw)

2) Websites-

- GeeksforGeeks-

This was useful in learning new scientific python modules which included- SciPy, Numpy, Pandas and Beautifulsoup.

- Additional Learning Sites-

1) https://numpy.org/doc/stable/user/absolute_beginners.html
2) https://www.w3schools.com/python/scipy/scipy_intro.php
3) https://docs.python.org/3/library/turtle.html
4) https://www.manim.community/
5) https://pypi.org/project/physics/

3) Graphics and Designs

1) Pysolver.in
2) www.canva.in
3) The_Effecient_Engineer
4) Motion Graphics- Manim.org
5) User-fed Algorithms- Researchgate.in
6) Voice Module and editor- Voice.AI and clipchamp