SCIENCE FORMULA CALCULATOR USING PYTHON PROGRAMMING LANGUAGE BY JATIN KOTARTHIL.

Submitted By,

Kotarthil Jatin Mohan.

MSc II Physics

A

Project report on

“Science Formula Calculator Using Python Programming Language by Jatin Kotarthil”

Submitted by

MR. KOTARTHIL JATIN MOHAN.

Guide by

Prof. P. V. Darade

In partial fulfillment for the award of the degree

Of

Master of Science

In

Physics

Kr. V. N. Naik Arts, Commerce and Science College,

Canada Corner, Nashik-02

Savitribai Phule Pune university, Pune 411 007

SEAT No:

Kr. V. N. NAIK SHIKSHAN PRASARAK SANSTHA'S

ARTS, COMMERCE & SCIENCE COLLEGE, CANADA CORNER, NASHIK 422002

DEPARTMENT OF PHYSICS

CERTIFICATE

This is to certify that this project report **"Science Formula Calculator Using Python Programming by Jatin Kotarthil"** is the bonafide work Of "**Mr. Kotarthil Jatin Mohan**" of M.Sc.-II (PHYSICS) during the Academic year 2022-2023 who carried out the project work under my Supervision.

PROJECT GUIDE	**HEAD OF DEPARTMENT**
(Prof. P. V. Darade)	**(Dr. V. G. Wagh)**

INTERNAL EXAMINER	**EXTERNAL EXAMINER**

DECLARATION

I, Kotarthil Jatin Mohan, hereby declare that the project Report entitled "Science Formula Calculator Using Python Programming Language by Jatin Kotarthil" Submitted in partial fulfillment of the requirements for Degree in M.Sc physics at Kr. V. N. Naik Arts, Commerce and Science College, is my original work. Under the Guidance of

P.V Darade sir

This is my original work.

I analyzed this work to enhance my programming Skills. I can do this work successfully.

ACKNOWLEDGMENT.

I feel great to present the project on "**Science Formula Calculator Using Python Programming Language**" In the first place, I would like to record my deep and sincere gratitude to Prof. P. V. Darade Sir for his Supervision, Advice, Guidance and Contribution. His Understanding, Encouraging a Personal Guidance has provided me a good basis for the present project.

I would like to express my special thanks of gratitude to our HOD Dr. V. G. Wagh Sir. I thank Prof. Bhalerao Sir for his time and guidance.

I am also grateful to all the other professors in the Department of Physics for their advice and knowledge passed on during the master's course. I extend my acknowledgment to my family, friends and to all those who have directly or indirectly help me in this project work.

Thank You!

You're sincerely

Mr. Jatin Kotarthil

ABSTRACT

In today's digital era, mobile applications have become an integral part of our daily lives, offering convenience and efficiency in various domains. This project focuses on the development of an Android application that aims to simplify the calculation of science formulas. The objective is to provide users with a versatile tool that enables them to perform complex scientific calculations effortlessly.

Through an intuitive user interface, the application offers a comprehensive collection of science formulas and equations across multiple disciplines. Users, including students, professionals, and science enthusiasts, can easily access and utilize these formulas to solve intricate problems, perform experiments, and validate hypotheses.

The project employs a systematic approach, utilizing modern programming languages, tools, and frameworks to develop a robust and user-friendly Android application. The system design emphasizes seamless navigation and effective organization of formulas, ensuring a streamlined user experience.

By creating this Android application, we aim to bridge the gap between theoretical knowledge and practical application in the field of science. This project report provides a comprehensive overview of the development process, methodology, system design, implementation details, testing procedures, and evaluation outcomes. The insights gained from this project contribute to the advancement of mobile technology in facilitating scientific computations and fostering a deeper understanding of scientific principles.

INDEX

CHAPATER 1:

1.1 INTRODUCTION

The rapid advancements in mobile technology have transformed the way we access and utilize information in our daily lives. With the increasing reliance on smartphones and the popularity of Android-based devices, there is a growing demand for mobile applications that provide convenient and efficient solutions for various tasks. In line with this trend, the focus of this final year project is the development of an Android application aimed at simplifying the calculation of science formulas.

The objective of this project is to create an intuitive and user-friendly application that enables students, professionals, and enthusiasts in the field of science to easily perform complex calculations and obtain accurate results. By leveraging the capabilities of modern smartphones, this application aims to enhance the learning and problem-solving experience for users, making science formulas accessible and comprehensible even to those with limited mathematical background.

Through this report, we will explore the development process, methodology, system design, implementation details, and testing procedures employed in the creation of this Android application. Additionally, we will evaluate the effectiveness and usability of the application based on user feedback and performance metrics.

By developing this application, we aim to contribute to the educational landscape by providing a practical tool that assists users in performing calculations, encourages curiosity and exploration, and fosters a deeper understanding of scientific concepts. The following sections will provide an in-depth analysis of the project, its outcomes, and potential avenues for future improvements.

By creating this Android application, we aim to bridge the gap between theoretical knowledge and practical application in the field of science. This project report provides a comprehensive overview of the development process, methodology, system design, implementation details, testing procedures, and evaluation outcomes. The insights gained from this project contribute to the advancement of mobile technology in facilitating scientific computations and fostering a deeper understanding of scientific principles.

Developing a mobile application for calculating science formulas offers several advantages and aligns with the evolving needs and preferences of users. Here are some reasons why this choice was made:

1. Accessibility and Convenience: Mobile applications provide easy access to information and tools on-the-go. By developing a mobile application for calculating science formulas, users can conveniently perform calculations anytime, anywhere, without the need for carrying physical calculators or accessing desktop applications.

2. Ubiquitous Adoption of Mobile Devices: The widespread use of smartphones and tablets has made mobile applications an integral part of our daily lives. By leveraging the popularity and ubiquity of mobile

devices, developing a mobile application ensures that it reaches a larger user base and caters to their preferences and habits.

3. Enhanced User Experience: Mobile applications offer a more intuitive and interactive user experience compared to traditional calculators or desktop applications. With touch-based interfaces, gestures, and multimedia capabilities, a mobile application can provide a visually appealing and engaging environment for users to interact with science formulas.

4. Integration of Additional Features: Unlike standalone calculators, mobile applications have the potential to integrate additional features and functionalities. For example, the application could include interactive simulations, visualizations, or reference materials that complement the calculations, enhancing the overall learning and problem-solving experience.

5. Educational Advancement: Developing a mobile application for calculating science formulas contributes to the advancement of educational tools and techniques. It promotes self-learning, encourages exploration, and facilitates the understanding of scientific concepts in a practical and engaging manner.

6. Adaptation to Changing Learning Environment: The modern education landscape is witnessing a shift towards digital resources and e-learning platforms. By developing a mobile application, you align with this trend and offer a valuable resource that can complement traditional classroom learning, online courses, or self-study.

Overall, developing a mobile application for calculating science formulas offers the advantages of accessibility, convenience, enhanced user experience, integration of additional features, educational advancement, and adaptation to the changing learning environment. These factors collectively make it a compelling choice to meet the needs of today's tech-savvy and mobile-oriented users.

1.2 LITERATURE REVIEW

Several mobile applications have been developed to assist users in performing scientific calculations. For example, "Physics Formula Calculator" by ATS Mobile Lab. provides a comprehensive collection of formulas and equations across various scientific disciplines. The application offers a user-friendly interface, allowing users to input variables and obtain results instantly. However, user feedback suggests a need for improved formula organization.

While existing applications have provided valuable contributions, there are several areas for improvement and further exploration. User feedback indicates the need for better formula organization, improved user interface design, and the inclusion of additional scientific disciplines.

CHAPATER 2:

METHODOLOGY

2.1 RESEARCH APPROACH:

To begin the project, a thorough research approach was adopted to understand the available options for developing cross-platform mobile applications and to select the most suitable framework. The research primarily focused on mobile application development frameworks and their compatibility with the Android platform. Key steps in this research approach included:

2.1.1 Literature Review:

- Conducted an extensive literature review to gain insights into different mobile application development frameworks.

- Explored research papers, online resources, and documentation related to mobile app development to understand the advantages and limitations of various frameworks.

2.1.2 Framework Evaluation:

- Analyzed different frameworks based on criteria such as cross-platform compatibility, user interface flexibility, community support, performance, and ease of use.

- Explored frameworks like React Native, Flutter, Xamarin, and Kivy, assessing their capabilities in developing Android applications.

2.1.3 Selection of Kivy:

- After careful evaluation, Kivy was selected as the framework of choice for developing the Android application.

- Considered Kivy's strong cross-platform support, its ability to develop rich and interactive user interfaces, and its compatibility with Python, a programming language known for its simplicity and readability.

- Reviewed community resources, online forums, and documentation to understand the learning curve and community support available for Kivy.

2.2 COMPONENTS:

I utilized specific programming languages, tools, and frameworks to develop your Android application for calculating science formulas. Here's an explanation of those components:

1. Programming Language: (Python)

Python is a versatile and high-level programming language known for its simplicity, readability, and extensive library support. It was created by Guido van Rossum and first released in 1991. Python emphasizes code readability through its clean and intuitive syntax, making it an excellent choice for beginners and experienced developers alike.

Key features and characteristics of Python include:

1. Easy-to-Read Syntax: Python uses a clean and straightforward syntax that emphasizes readability and reduces the need for excessive punctuation or complex syntax rules. It uses indentation (whitespace) to define code blocks, making it highly readable and visually appealing.

2. Interpreted Language: Python is an interpreted language, meaning that it does not require explicit compilation. The Python interpreter reads and executes the code line by line, making it suitable for rapid prototyping, testing, and interactive development.

3. Cross-platform Compatibility: Python is available on various operating systems, including Windows, macOS, Linux, and others. This cross-platform compatibility allows developers to write code on one platform and run it on multiple platforms without modification.

4. Extensive Standard Library: Python comes with a comprehensive standard library that provides a wide range of pre-built modules and functions, offering solutions for tasks such as file I/O, networking, regular expressions, data manipulation, and more. This extensive library saves development time and encourages code reuse.

5. Third-party Libraries and Frameworks: Python has a vast ecosystem of third-party libraries and frameworks, enabling developers to leverage existing tools and packages to enhance their projects. Examples include NumPy for scientific computing, Django for web development, TensorFlow for machine learning, and Matplotlib for data visualization.

6. Object-Oriented Programming (OOP) Support: Python supports object-oriented programming, allowing developers to create and use classes, objects, and inheritance to structure and organize their code. This OOP support promotes code modularity, reusability, and maintainability.

Python finds applications in various domains, including web development, data analysis, scientific computing, machine learning, artificial intelligence, scripting, and automation. Its versatility, ease of use, and large developer community make it a popular choice for both beginners and experienced developers seeking to build a wide range of applications.

2. Tool : (Anaconda)

Anaconda is a popular distribution of the Python programming language specifically designed for data science and scientific computing. It aims to simplify the installation and management of Python and related libraries by providing a comprehensive package management system. Here's an explanation of Anaconda and its key features:

1. Package Management: Anaconda includes a powerful package manager called "conda." Conda allows you to easily install, update, and manage Python packages and dependencies. It provides access to a vast collection of pre-built libraries, making it convenient for data scientists and developers working with data analysis, machine learning, and scientific computing.

2. Cross-Platform Compatibility: Anaconda supports multiple operating systems, including Windows, macOS, and Linux. This cross-platform compatibility ensures that your Python projects and packages work consistently across different environments.

3. Environment Management: Anaconda allows you to create isolated Python environments, known as "conda environments." Each environment can have its own set of Python packages and dependencies, enabling you to maintain separate and controlled development environments for different projects. This helps to avoid conflicts between package versions and provides reproducibility.

4. Integrated Development Environment (IDE): Anaconda includes the option to install the Anaconda Navigator, a graphical user interface (GUI) that provides an integrated development environment for Python.

The Navigator simplifies package installation, environment management, and project organization through a user-friendly interface.

5. Data Science Libraries: Anaconda comes bundled with popular data science libraries such as NumPy, Pandas, Matplotlib, scikit-learn, and Jupyter Notebook. These libraries are pre-installed and optimized for compatibility, making it easy to start working on data analysis, visualization, and machine learning tasks right out of the box.

Overall, Anaconda provides a streamlined and comprehensive environment for Python-based data science and scientific computing. It simplifies package management, offers cross-platform compatibility, and includes essential libraries, making it a popular choice for data scientists and developers in these domains.

3. Framework: (Kivy)

Kivy is an open-source Python framework for developing cross-platform applications, particularly focused on creating user interfaces (UIs) and user experiences (UX). It is designed to enable the development of applications that run on various platforms, including Windows, macOS, Linux, Android, and iOS. Here's an explanation of Kivy and its key features:

1. Cross-Platform Compatibility: Kivy's main strength lies in its ability to create applications that work seamlessly across different operating systems and devices. It achieves this by leveraging platform-specific implementations and providing a consistent API for developers.

2. Python-based: Kivy is built on top of Python, which offers several advantages. Python is a high-level, easy-to-read programming language known for its simplicity and extensive libraries. Developers familiar with Python can utilize their existing knowledge to build applications using Kivy.

3. Rapid Development: Kivy emphasizes rapid application development through its concise and expressive syntax. It provides a domain-specific language (DSL) called the Kivy Language (KV), which allows developers to describe user interfaces declaratively, reducing the need for extensive coding. This declarative approach helps streamline the development process and makes UI design more intuitive.

4. Rich User Interfaces: Kivy enables the creation of visually appealing and interactive user interfaces. It offers a wide range of UI elements, including buttons, labels, text inputs, dropdown menus, and more. These elements can be customized using various styling options, such as

colors, fonts, and layouts, to create unique and engaging user experiences.

5. Multi-Touch and Gestures: Kivy supports multi-touch and gestures, allowing developers to create applications that respond to touch interactions. This feature is particularly beneficial for mobile and touch-enabled devices, enabling the development of intuitive and interactive applications.

6. Hardware Acceleration: Kivy leverages hardware acceleration whenever possible to provide smooth and performant graphics rendering. It takes advantage of the graphics processing unit (GPU) to offload computation and rendering tasks, resulting in improved performance and responsiveness.

Overall, Kivy is a versatile and powerful framework for creating cross-platform applications with rich and interactive user interfaces. Its cross-platform compatibility, Python integration, rapid development approach, and extensive UI capabilities make it a popular choice for developers aiming to build visually appealing and user-friendly applications.

4. Libraries and Modules:

1. **NumPy:** NumPy is a Python library that provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions. It may have been used to handle scientific calculations and numerical operations in your application.
2. **Matplotlib:** Matplotlib is a plotting library in Python that enables the creation of various types of visualizations, including graphs, charts, and plots. You may have incorporated Matplotlib to display data or generate visual representations of scientific formulas or calculations in your application.
3. **Pandas:** Pandas is a popular Python library for data manipulation and analysis. It provides data structures and functions to efficiently handle and process structured data. You might have utilized Pandas to manage and manipulate scientific data within your application.
4. **"Kivy":** This is the main Physics Module.
5. **"math":** This is a Pythons built in Module for Mathematical Function

CHAPATER 3:

APPLICATION INDEX

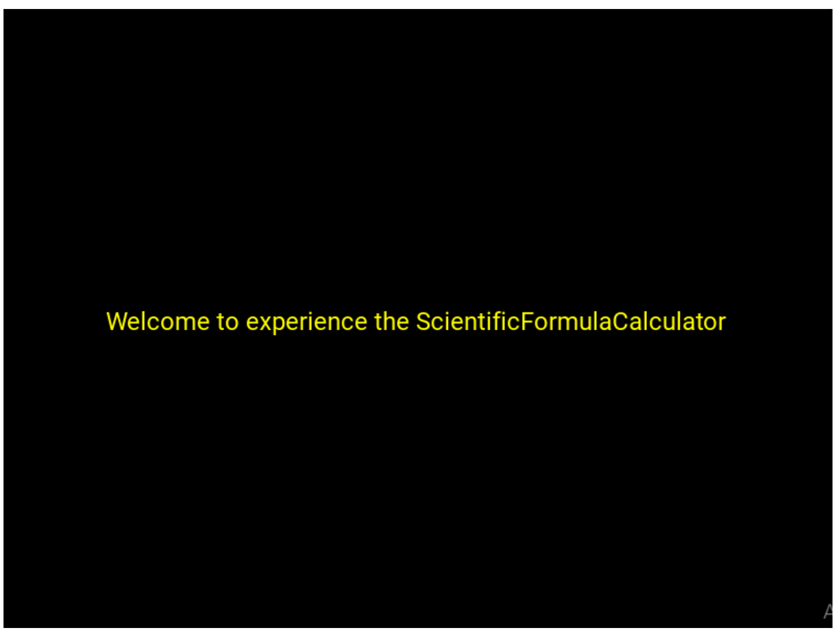
1. Splash Page
2. Main page
    - 2.1 Physics page
    - 2.2 Chemistry Page
    - 2.3 Math's Page
    - 2.4 Calculator
3. Physics page
    - 3.1 Classical Mechanics
    - 3.2 Thermodynamics
    - 3.3 Electrodynamics
    - 3.4 Optics
    - 3.5 Elasticity
    - 3.6 Astrophysics
4. Chemistry Page
    - 4.1 Solutions And Co. Properties.
5. Math's page
    - 5.1 Area
    - 5.2 Perimeter
    - 5.3 Expansions
    - 5.4 Trigonometric Value
    - 5.5 Trigonometric Value (Inverse)
    - 5.6 Logarithms
6. Calculator

CHAPATER 4:

SYSTEM DESIGN

The application comprises six pages, starting with a Splash page that displays for 3 seconds as a welcome screen. Following the Splash page, the main page appears, featuring four components: physics, chemistry, mathematics, and a regular calculator. Each subject directs the user to its respective syllabus, which is individually described below with all pages.

1. Splash page



On Screen Image.

The splash page blinks for 3 seconds when the application is opened. It welcomes the user by displaying a message. The splash page serves as the initial welcome page of the application.

CODE WITH EXPLAINATION:

```
class SplashPage(Screen):
    def __init__(self, **kwargs):
        super(SplashPage, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.label = Label(text='Welcome to experience the
ScientificFormulaCalculator', font_size=24, color=[1, 1, 0, 1])
        self.layout.add_widget(self.label)
        self.add_widget(self.layout)
        Clock.schedule_once(self.go_to_main_page, 10)

    def go_to_main_page(self, dt):
        self.manager.current = 'main'
```

The code provided represents a class called `SplashPage` which inherits from the `Screen` class in Kivy. This class defines the behavior and layout of the splash page in your application. Here's a breakdown of the code:

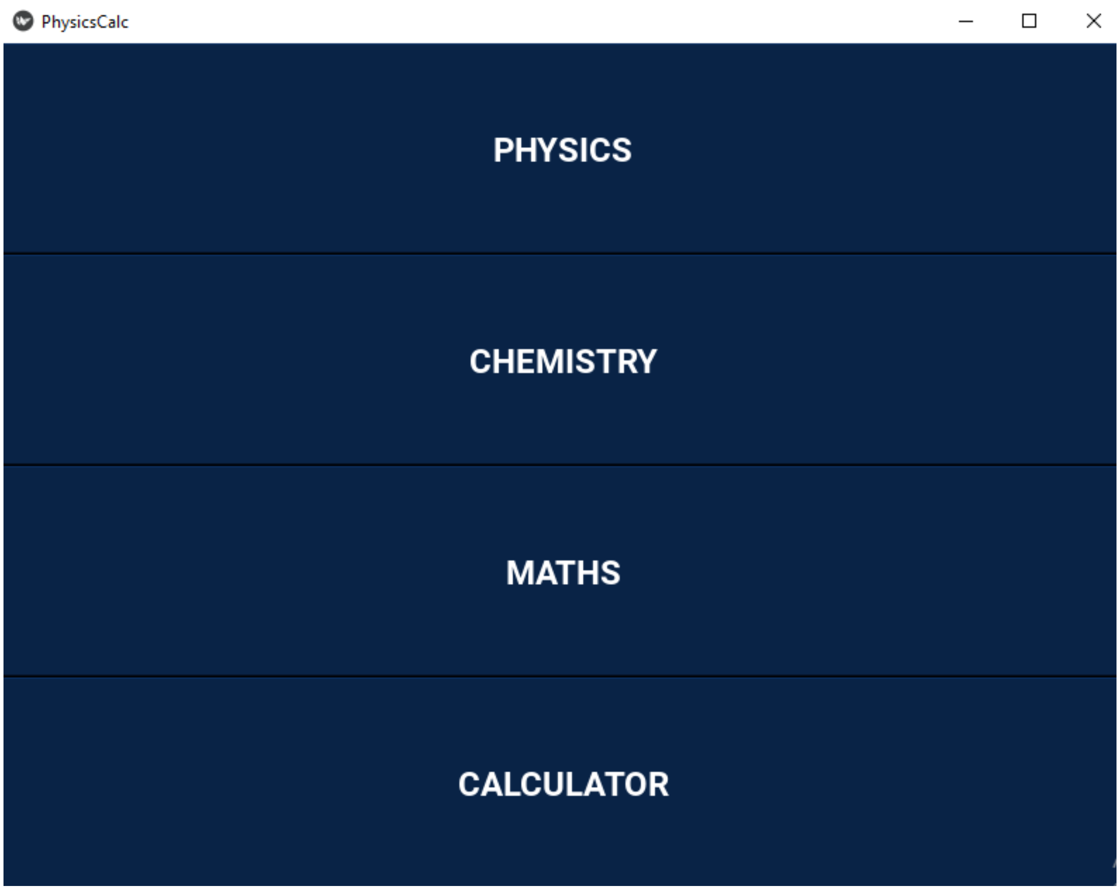
1. The `__init__` method is the constructor of the `SplashPage` class. It initializes the layout, creates a label widget with a welcome message, and adds the label to the layout using a `BoxLayout` with a vertical orientation.

2. The `Clock.schedule_once` method is used to schedule a function call (`go_to_main_page`) after a specified delay of 3 seconds. This function is responsible for transitioning to the main page of the application.

3. The `go_to_main_page` method is called after the specified delay. It sets the current screen of the `ScreenManager` (assuming the `ScreenManager` is managing the screens) to 'main', which triggers the transition to the main page.

Overall, the `SplashPage` class creates a simple splash page with a welcome message and automatically navigates to the main page after 3 seconds.

2. Main page :



On Screen Image

The main page consists of four main topics or components named PHYSICS, CHEMISTRY, MATHS, and CALCULATOR. Upon selecting a component, the user is directed to the corresponding topic or module page of the selected component's syllabus.

CODE WITH EXPLAINATION:

```
class MainPage(Screen):

    def __init__(self, **kwargs):

        super(MainPage, self).__init__(**kwargs)

        self.layout = BoxLayout(orientation='vertical')

        self.physics_button = Button(text='PHYSICS',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],

                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.physics_press)

        self.chemistry_button = Button(text='CHEMISTRY',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],

                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.chemistry_press)

        self.math_button = Button(text='MATHS', font_size=24,
size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],

                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.math_press)

        self.calculator_button = Button(text='CALCULATOR',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],

                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.calculator_press)

        self.layout.add_widget(self.physics_button)

        self.layout.add_widget(self.chemistry_button)

        self.layout.add_widget(self.math_button)

        self.layout.add_widget(self.calculator_button)

        self.add_widget(self.layout)
```

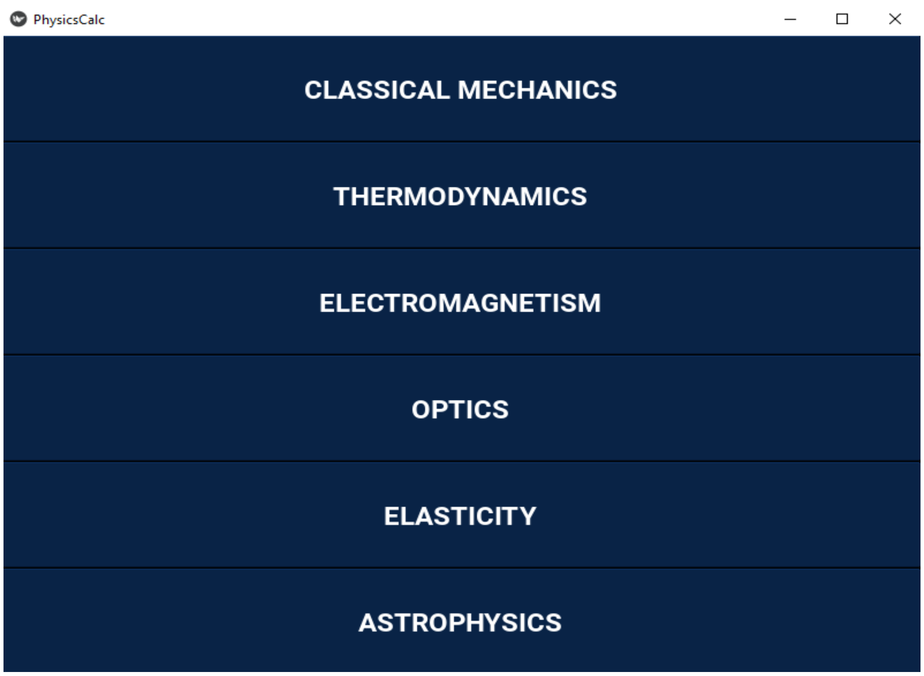
```
def physics_press(self, instance):
    self.manager.current = 'Phy'
def chemistry_press(self, instance):
    self.manager.current = 'chem'
def math_press(self, instance):
    self.manager.current = 'math'
def calculator_press(self, instance):
    self.manager.current = 'Calc'
```

The code provided represents a class called `MainPage` which inherits from the `Screen` class in Kivy. This class defines the behavior and layout of the main page in your application. Here's a breakdown of the code:

1. The `__init__` method is the constructor of the `MainPage` class. It initializes the layout as a vertical `BoxLayout` and creates four buttons for physics, chemistry, mathematics, and calculator components. Each button is styled with specific properties such as text, font size, size hint, boldness, color, background color, and assigns an `on_press` event handler to handle button presses.
2. The `physics_press`, `chemistry_press`, `math_press`, and `calculator_press` methods are the event handler functions for the respective buttons. When a button is pressed, these methods change the current screen of the `ScreenManager` to the corresponding screen (`Phy`, `chem`, `math`, `Calc`) using the `manager.current` attribute.
3. The buttons are added to the layout using the `add_widget` method, and the layout is added to the `MainPage` using `add_widget(self.layout)`.

Overall, the `MainPage` class creates a main page with four buttons representing different components. Pressing each button transitions to a specific screen corresponding to the chosen component.

3. Physics page:



On Screen Image

After selecting the PHYSICS button, the user is directed to the Physics module or Physics syllabus. The syllabus consists of specific formula modules such as

Classical Mechanics,

Thermodynamics,

Electromagnetism,

Optics,

Elasticity,

And Astrophysics.

The user can select a particular module to be directed to the corresponding topic's formulas.

CODE WITH EXPLAINATION:

```
class PhysicsCalc(Screen):

    def __init__(self, **kwargs):

        super(PhysicsCalc, self).__init__(**kwargs)

        self.layout = BoxLayout(orientation='vertical')

        self.classical_button = Button(text='CLASSICAL
MECHANICS', font_size=24, size_hint=(1, 0.3), bold=True,
color=[1, 1, 1, 1],

                                       background_color=[0.1, 0.4,
0.8, 1], on_press=self.classical_press)

        self.thermo_button = Button(text='THERMODYNAMICS',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],

                                       background_color=[0.1, 0.4,
0.8, 1], on_press=self.thermo_press)

        self.electromag_button = Button(text='ELECTROMAGNETISM',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],

                                       background_color=[0.1, 0.4,
0.8, 1], on_press=self.electromag_press)

        self.optics_button = Button(text='OPTICS', font_size=24,
size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],

                                       background_color=[0.1, 0.4,
0.8, 1], on_press=self.optics_press)

        self.elasticity_button = Button(text='ELASTICITY',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
```

```
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.elasticity_press)
        self.astro_button = Button(text='ASTROPHYSICS',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.astro_press)



        self.layout.add_widget(self.classical_button)
        self.layout.add_widget(self.thermo_button)
        self.layout.add_widget(self.electromag_button)
        self.layout.add_widget(self.optics_button)
        self.layout.add_widget(self.elasticity_button)
        self.layout.add_widget(self.astro_button)
        self.add_widget(self.layout)

    def classical_press(self, instance):
        self.manager.current = 'classical'
    def thermo_press(self, instance):
        self.manager.current = 'thermo'
    def electromag_press(self, instance):
        self.manager.current = 'electromag'
    def optics_press(self, instance):
        self.manager.current = 'optics'
    def elasticity_press(self, instance):
        self.manager.current = 'elast'
```

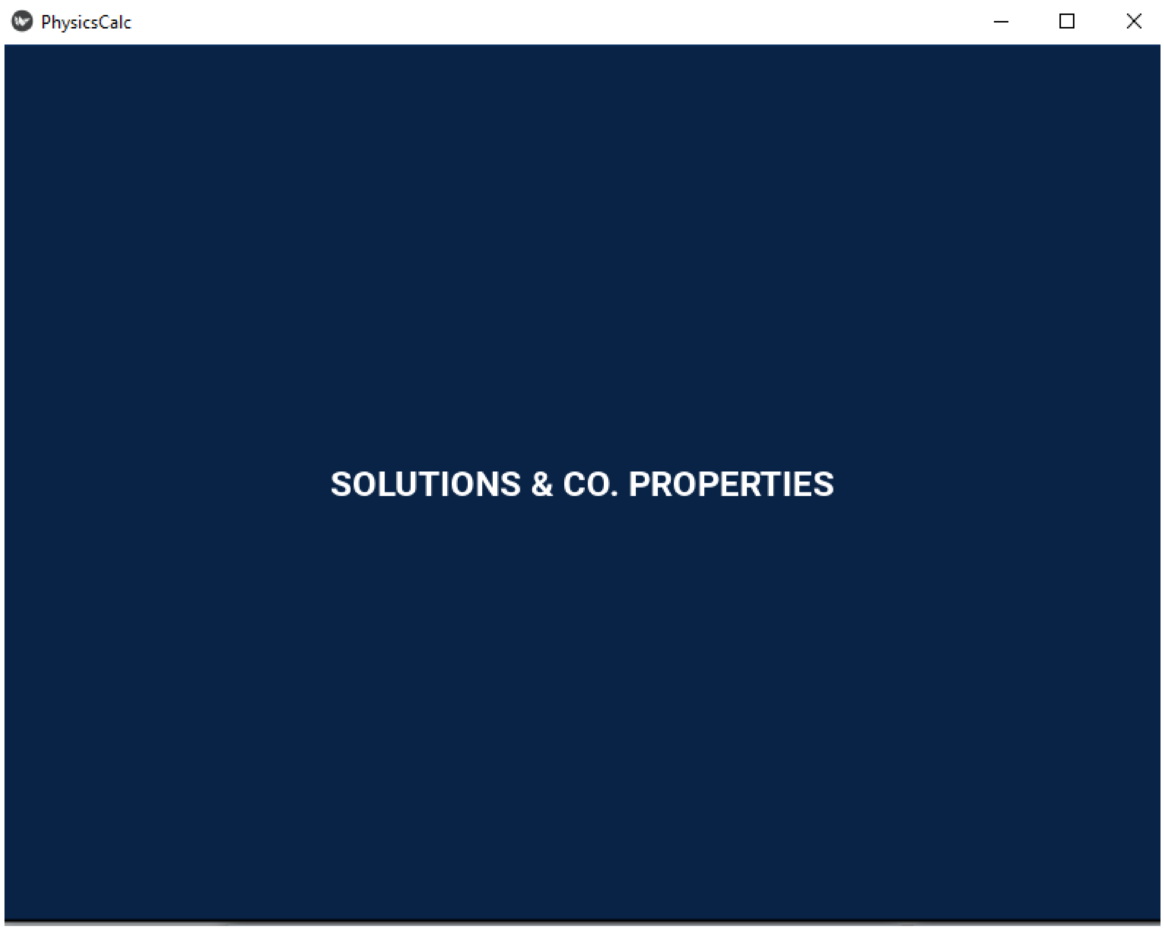
```
    def astro_press(self, instance):
        self.manager.current = 'astro'
```

The code provided represents a class called `PhysicsCalc` which inherits from the `Screen` class in Kivy. This class defines the behavior and layout of the physics component page in your application. Here's a breakdown of the code:

1. The `__init__` method is the constructor of the `PhysicsCalc` class. It initializes the layout as a vertical `BoxLayout` and creates six buttons for different areas of physics: classical mechanics, thermodynamics, electromagnetism, optics, elasticity, and astrophysics. Each button is styled with specific properties such as text, font size, size hint, boldness, color, background color, and assigns an `on_press` event handler to handle button presses.
2. The `classical_press`, `thermo_press`, `electromag_press`, `optics_press`, `elasticity_press`, and `astro_press` methods are the event handler functions for the respective buttons. When a button is pressed, these methods change the current screen of the `ScreenManager` to the corresponding screen (`classical`, `thermo`, `electromag`, `optics`, `elast`, `astro`) using the `manager.current` attribute.
3. The buttons are added to the layout using the `add_widget` method, and the layout is added to the `PhysicsCalc` using `add_widget(self.layout)`.

Overall, the `PhysicsCalc` class creates a page for the physics component with buttons representing different areas of physics. Pressing each button transitions to a specific screen corresponding to the chosen physics area

. 4. Chemistry page:



On Screen Image

After selecting the CHEMISTRY button, the user is directed to the Chemistry module or Chemistry syllabus. The syllabus consists of Solution and Co. Properties formula modules.

The user can select a particular module to be directed to the corresponding topic's formulas.

CODE WITH EXPLAINATION:

```
class ChemistryCalc(Screen):
    def __init__(self, **kwargs):
        super(ChemistryCalc, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.sp_button = Button(text='SOLUTIONS & CO. PROPERTIES', font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4, 0.8, 1], on_press=self.sp_press)
        self.layout.add_widget(self.sp_button)
        self.add_widget(self.layout)
    def sp_press(self, instance):
        self.manager.current = 'sp'
```

The code provided represents a class called `ChemistryCalc` which inherits from the `Screen` class in Kivy. This class defines the behavior and layout of the chemistry component page in your application. Here's a breakdown of the code:

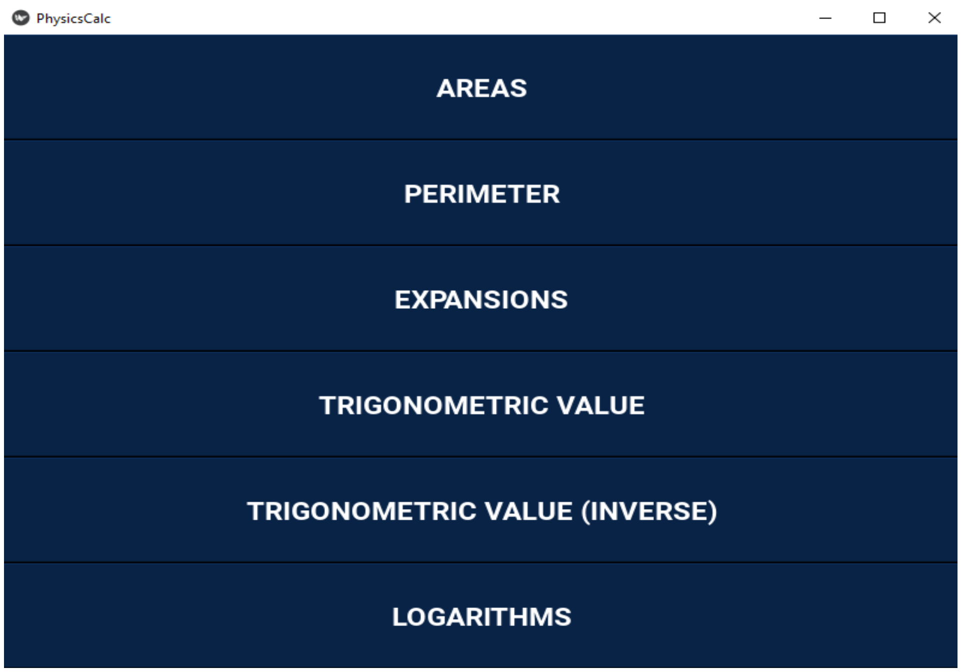
1. The `__init__` method is the constructor of the `ChemistryCalc` class. It initializes the layout as a vertical `BoxLayout` and creates a button for the solutions and co. properties component of chemistry. The button is styled with specific properties such as

text, font size, size hint, boldness, color, background color, and assigns an `on_press` event handler to handle button presses.

2. The `sp_press` method is the event handler function for the button. When the button is pressed, this method changes the current screen of the `ScreenManager` to the 'sp' screen using the `manager.current` attribute.
3. The button is added to the layout using the `add_widget` method, and the layout is added to the `ChemistryCalc` using `add_widget(self.layout)`.

Overall, the `ChemistryCalc` class creates a page for the chemistry component with a button representing the solutions and co. properties. Pressing the button transitions to the 'sp' screen corresponding to the chosen chemistry area.

. 5. Math's page:



On Screen Image

After selecting the MATHS button, the user is directed to the Math's module or Math's syllabus. The syllabus consists of specific formula modules such as

Area,

Perimeter,

Expansion,

Trigonometric Value,

Trigonometric Value (Inverse),

Logarithms

The user can select a particular module to be directed to the corresponding topic's formulas.

CODE WITH EXPLAINATION:

```
class MathCalc(Screen):
    def __init__(self, **kwargs):
        super(MathCalc, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.area_button = Button(text='AREAS', font_size=24,
size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.area_press)
        self.perimeter_button = Button(text='PERIMETER',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.perimeter_press)
        self.expansions_button = Button(text='EXPANSIONS',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.expansions_press)
        self.trigono_button = Button(text='TRIGONOMETRIC VALUE',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.trigono_press)
        self.itrigono_button = Button(text='TRIGONOMETRIC VALUE
(INVERSE)', font_size=24, size_hint=(1, 0.3), bold=True,
color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.itrigono_press)
        self.log_button = Button(text='LOGARITHMS',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
```

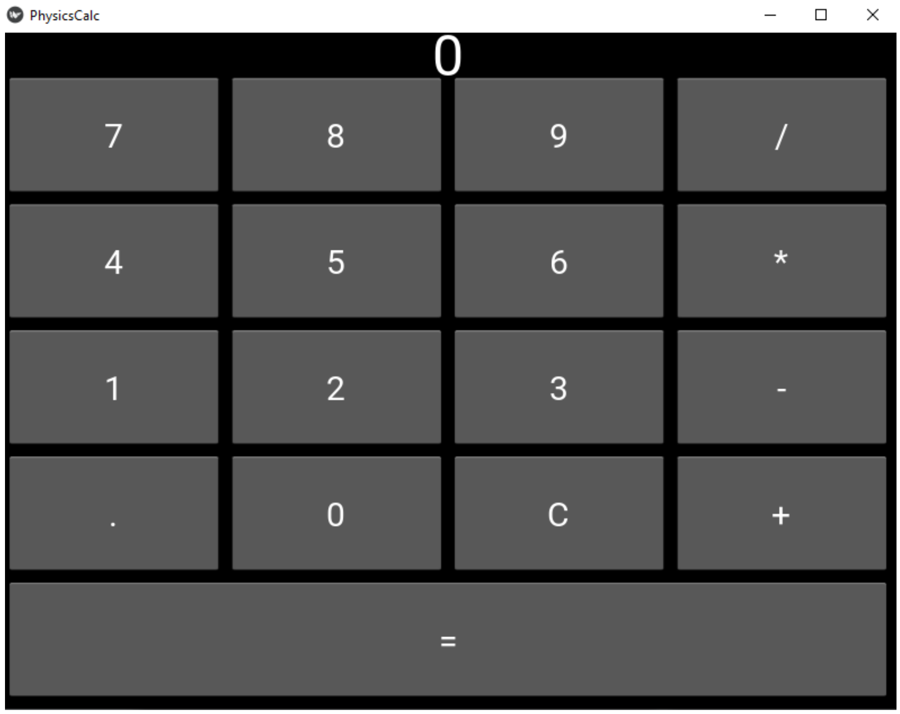
```
                                      background_color=[0.1, 0.4,
0.8, 1], on_press=self.log_press)
        self.layout.add_widget(self.area_button)
        self.layout.add_widget(self.perimeter_button)
        self.layout.add_widget(self.expansions_button)
        self.layout.add_widget(self.trigono_button)
        self.layout.add_widget(self.itrigono_button)
        self.layout.add_widget(self.log_button)
        self.add_widget(self.layout)
    def area_press(self, instance):
        self.manager.current = 'area'
    def perimeter_press(self, instance):
        self.manager.current = 'peri'
    def expansions_press(self, instance):
        self.manager.current = 'exp'
    def trigono_press(self, instance):
        self.manager.current = 'tri'
    def itrigono_press(self, instance):
        self.manager.current = 'itri'
    def log_press(self, instance):
        self.manager.current = 'log'
```

The code provided represents a class called `MathCalc` which inherits from the `Screen` class in Kivy. This class defines the behavior and layout of the math component page in your application. Here's a breakdown of the code:

1. The `__init__` method is the constructor of the `MathCalc` class. It initializes the layout as a vertical `BoxLayout` and creates six buttons for different areas of math: areas, perimeter, expansions, trigonometric values, inverse trigonometric values, and logarithms. Each button is styled with specific properties such as text, font size, size hint, boldness, color, background color, and assigns an `on_press` event handler to handle button presses.
2. The `area_press`, `perimeter_press`, `expansions_press`, `trigono_press`, `itrigono_press`, and `log_press` methods are the event handler functions for the respective buttons. When a button is pressed, these methods change the current screen of the `ScreenManager` to the corresponding screen (`area`, `peri`, `exp`, `tri`, `itri`, `log`) using the `manager.current` attribute.
3. The buttons are added to the layout using the `add_widget` method, and the layout is added to the `MathCalc` using `add_widget(self.layout)`.

Overall, the `MathCalc` class creates a page for the math component with buttons representing different areas of math. Pressing each button transitions to a specific screen corresponding to the chosen math area.

6. Calculator page:



On Screen Image

CODE WITH EXPLAINATION:

```
class Calculator(Screen):

    def __init__(self, **kwargs):

        super(Calculator, self).__init__(**kwargs)

        # Create a BoxLayout to hold the UI elements

        layout = BoxLayout(orientation='vertical', spacing=10,
padding=10)

        # Create a label to display the calculator input/output

        self.display = Label(text='0', font_size=50,
halign='right', size_hint=(1, 0.2))
```

```
        layout.add_widget(self.display)
        # Create buttons for the calculator

            buttons = [
            ['7', '8', '9', '/'],
            ['4', '5', '6', '*'],
            ['1', '2', '3', '-'],
            ['.', '0', 'C', '+'],
            ['=',]
        ]
        # Add the buttons to the layout
        for row in buttons:
            row_layout = BoxLayout(spacing=10)
            for label in row:
                button = Button(text=label, font_size=30,
size_hint=(0.2, 1))
                button.bind(on_press=self.on_button_press)
                row_layout.add_widget(button)
            layout.add_widget(row_layout)
        self.add_widget(layout)
    def on_button_press(self, instance):
        # Get the current input/output from the display
        text = self.display.text
        # Handle the different button presses
        if instance.text == 'C':
```

```
            # Clear the display
            self.display.text = '0'
        elif instance.text == '=':
            # Evaluate the expression and display the result
            try:
                result = str(eval(text))
                self.display.text = result
            except:
                self.display.text = 'Error'
        else:
            # Append the button label to the display
            if text == '0':
                self.display.text = instance.text
            else:
                self.display.text = text + instance.text
```

The code represents a class called `Calculator` which inherits from the `Screen` class in Kivy. This class defines the behavior and layout of the calculator component in your application. Here's a breakdown of the code:

1. The `__init__` method is the constructor of the `Calculator` class. It creates a vertical `BoxLayout` called `layout` to hold the UI elements. Within this layout, it creates a `Label` called `display` to show the calculator input/output. The label is styled with a large font size, aligned to the right, and takes up 20% of the available height.

2. Next, it defines a list called `buttons` that represents the buttons for the calculator. Each inner list represents a row of buttons, and each label in the row represents a button's text.
3. The buttons are added to the layout by iterating over the `buttons` list. For each row, it creates a horizontal `BoxLayout` called `row_layout`, and for each label, it creates a `Button` with the corresponding text, font size, and size hint. The `on_button_press` method is bound to the `on_press` event of each button.
4. The `on_button_press` method is the event handler for button presses. It retrieves the current input/output from the display label. Then, based on the button pressed, it performs different actions: clearing the display if the button is 'C', evaluating the expression and displaying the result if the button is '=', or appending the button label to the display otherwise.
5. Finally, the layout is added to the `Calculator` screen using `add_widget(layout)`.

Overall, the `Calculator` class creates a calculator component with a display label and a grid of buttons. Pressing the buttons updates the display label accordingly, allowing the user to perform calculations.

CHAPATER 5:
COMPLETE REQUIRED CODE

```
import kivy
import math
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
from kivy.uix.screenmanager import Screen
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.screenmanager import ScreenManager, Screen
from kivy.clock import Clock
from kivy.core.window import Window
from kivy.uix.behaviors import ButtonBehavior
from kivy.uix.image import Image


class SplashPage(Screen):
    def __init__(self, **kwargs):
        super(SplashPage, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.label = Label(text='Welcome to experience the
ScientificFormulaCalculator', font_size=24, color=[1, 1, 0, 1])
        self.layout.add_widget(self.label)
        self.add_widget(self.layout)
        Clock.schedule_once(self.go_to_main_page, 10)

    def go_to_main_page(self, dt):
        self.manager.current = 'main'


class MainPage(Screen):
    def __init__(self, **kwargs):
        super(MainPage, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.physics_button = Button(text='PHYSICS',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.physics_press)
```

```
        self.chemistry_button = Button(text='CHEMISTRY',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.chemistry_press)
        self.math_button = Button(text='MATHS', font_size=24,
size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.math_press)
        self.calculator_button = Button(text='CALCULATOR',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.calculator_press)


        self.layout.add_widget(self.physics_button)
        self.layout.add_widget(self.chemistry_button)
        self.layout.add_widget(self.math_button)
        self.layout.add_widget(self.calculator_button)
        self.add_widget(self.layout)

    def physics_press(self, instance):
        self.manager.current = 'Phy'
    def chemistry_press(self, instance):
        self.manager.current = 'chem'
    def math_press(self, instance):
        self.manager.current = 'math'
    def calculator_press(self, instance):
        self.manager.current = 'Calc'

class ChemistryCalc(Screen):
    def __init__(self, **kwargs):
        super(ChemistryCalc, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.sp_button = Button(text='SOLUTIONS & CO.
PROPERTIES', font_size=24, size_hint=(1, 0.3), bold=True,
color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.sp_press)

        self.layout.add_widget(self.sp_button)
        self.add_widget(self.layout)
    def sp_press(self, instance):
        self.manager.current = 'sp'
```

```
class Sols(Screen):
    def __init__(self, **kwargs):
        super(Sols, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.add_widget(self.layout)

        # Add buttons for each formula
        self.layout.add_widget(Button(text='PERCENTAGE BY WT. OF
SOLUTION [%WT= (Ws/Wsol)*100]', on_press=self.calculate_wtsol))
        self.layout.add_widget(Button(text='MOLE FRACTION OF
SOLUTION [Msol=Mole/(Ms+Msv)]', on_press=self.calculate_molfrac))
        self.layout.add_widget(Button(text='MOLARITY
[M=Wsolute/MWsolute]', on_press=self.calculate_molarity))
        self.layout.add_widget(Button(text='MOLALITY
[M=Wsolute/MWsolute*Wsolvent]',
on_press=self.calculate_molality))
        self.layout.add_widget(Button(text='NORMALITY
[N=Gsolute/Vsolution]', on_press=self.calculate_normality))

        # Create a label to display the result
        self.result_label = Label(text='Result: ')

        # Add the label to the grid
        self.layout.add_widget(self.result_label)

        # Formula and caluculation.
    def calculate_normality(self,instance):
        Gsolute_input=TextInput(text='0', multiline=False)
        Vsolution_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            Gsolute=float(Gsolute_input.text)
            Vsolution=float(Vsolution_input.text)

            w=Gsolute/Vsolution
            self.result_label.text = f'Result: Normality (N) of
solution is {w}. '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
```

```
        self.layout.add_widget(Label(text='Weight of Solute in
g/dm^3 of Solution (Wsolute) '))
        self.layout.add_widget(Gsolute_input)
        self.layout.add_widget(Label(text='Molecular weight of
Solute (MWsolute) '))
        self.layout.add_widget(Vsolution_input)
        self.layout.add_widget(calc_button)

    def calculate_molality(self,instance):
        Wsolute_input=TextInput(text='0', multiline=False)
        MWsolute_input=TextInput(text='0', multiline=False)
        Wsolvent_input=TextInput(text='0', multiline=False)
        def calculate(instance):
            Wsolute=float(Wsolute_input.text)
            MWsolute=float(MWsolute_input.text)
            Wsolvent=float(Wsolvent_input.text)
            w=Wsolute/(MWsolute*Wsolvent)
            self.result_label.text = f'Result: Molality (M) of
solution is {w}. '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Weight of Solute of
Solution (Wsolute) '))
        self.layout.add_widget(Wsolute_input)
        self.layout.add_widget(Label(text='Molecular weight of
Solute (MWsolute) '))
        self.layout.add_widget(MWsolute_input)
        self.layout.add_widget(Label(text='Weight of Solvent
(Wsolvent) '))
        self.layout.add_widget(Wsolvent_input)
        self.layout.add_widget(calc_button)

    def calculate_molarity(self,instance):
        Wsolute_input=TextInput(text='0', multiline=False)
        MWsolute_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            Wsolute=float(Wsolute_input.text)
            MWsolute=float(MWsolute_input.text)

            w=Wsolute/MWsolute
```

```
            self.result_label.text = f'Result: Molarity (M) of
solution is {w}. '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Weight of Solute in
g/dm^3 of Solution (Wsolute) '))
        self.layout.add_widget(Wsolute_input)
        self.layout.add_widget(Label(text='Molecular weight of
Solute (MWsolute) '))
        self.layout.add_widget(MWsolute_input)
        self.layout.add_widget(calc_button)

    def calculate_molfrac(self,instance):
        Mole_input=TextInput(text='0', multiline=False)
        Ms_input=TextInput(text='0', multiline=False)
        Msv_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            Mole=float(Mole_input.text)
            Ms=float(Ms_input.text)
            Msv=float(Msv_input.text)
            w=(Mole)/(Ms+Msv)
            self.result_label.text = f'Result: Mole Fraction of a
component of a Solution is {w}. '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Mole of Solute (Mole)
'))
        self.layout.add_widget(Mole_input)
        self.layout.add_widget(Label(text='Total Number of Moles
of Solute (Ms) '))
        self.layout.add_widget(Ms_input)
        self.layout.add_widget(Label(text='Total Number of Moles
of Solvent (Msv) '))
        self.layout.add_widget(Msv_input)
        self.layout.add_widget(calc_button)

    def calculate_wtsol(self,instance):
        Ws_input=TextInput(text='0', multiline=False)
```

```
        Wsol_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            Ws=float(Ws_input.text)
            Wsol=float(Wsol_input.text)
            w=(Ws/Wsol)*100
            self.result_label.text = f'Result: Percentage by
Weight of a Solution is {w}. '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Weight of Solute (Ws)
'))
        self.layout.add_widget(Ws_input)
        self.layout.add_widget(Label(text='Weight of Solution
(Wsol) '))
        self.layout.add_widget(Wsol_input)
        self.layout.add_widget(calc_button)

class MathCalc(Screen):
    def __init__(self, **kwargs):
        super(MathCalc, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.area_button = Button(text='AREAS', font_size=24,
size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.area_press)
        self.perimeter_button = Button(text='PERIMETER',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.perimeter_press)
        self.expansions_button = Button(text='EXPANSIONS',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.expansions_press)
        self.trigono_button = Button(text='TRIGONOMETRIC VALUE',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                     background_color=[0.1, 0.4,
0.8, 1], on_press=self.trigono_press)
        self.itrigono_button = Button(text='TRIGONOMETRIC VALUE
(INVERSE)', font_size=24, size_hint=(1, 0.3), bold=True,
color=[1, 1, 1, 1],
```

```
                                          background_color=[0.1, 0.4,
0.8, 1], on_press=self.itrigono_press)
        self.log_button = Button(text='LOGARITHMS', font_size=24,
size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                          background_color=[0.1, 0.4,
0.8, 1], on_press=self.log_press)


        self.layout.add_widget(self.area_button)
        self.layout.add_widget(self.perimeter_button)
        self.layout.add_widget(self.expansions_button)
        self.layout.add_widget(self.trigono_button)
        self.layout.add_widget(self.itrigono_button)
        self.layout.add_widget(self.log_button)
        self.add_widget(self.layout)

    def area_press(self, instance):
        self.manager.current = 'area'
    def perimeter_press(self, instance):
        self.manager.current = 'peri'
    def expansions_press(self, instance):
        self.manager.current = 'exp'
    def trigono_press(self, instance):
        self.manager.current = 'tri'
    def itrigono_press(self, instance):
        self.manager.current = 'itri'
    def log_press(self, instance):
        self.manager.current = 'log'

class Logarithms(Screen):
    def __init__(self, **kwargs):
        super(Logarithms, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.add_widget(self.layout)

         # Add buttons for each formula
        self.layout.add_widget(Button(text=' LOGARITHM ',
on_press=self.calculate_log))
        self.layout.add_widget(Button(text=' ANTI-LOGARITHM ',
on_press=self.calculate_antilog))

        # Create a label to display the result
        self.result_label = Label(text='Result: ')
```

```
        # Add the label to the grid
        self.layout.add_widget(self.result_label)

        # Formula and caluculations

    def calculate_antilog(self,instance):
        log_value_input=TextInput(text='0', multiline=False)
        base_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            log_value=float(log_value_input.text)
            base=float(base_input.text)
            antilog = math.pow(base, log_value)
            self.result_label.text = f'Result: The anti-logarithm 
of {log_value} to the base {base} is {antilog}. '

        calc_button = Button(text='Calculate', 
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Enter the logarithm 
value: '))
        self.layout.add_widget(log_value_input)
        self.layout.add_widget(Label(text='Enter the base of the 
logarithm:  '))
        self.layout.add_widget(base_input)
        self.layout.add_widget(calc_button)

    def calculate_log(self,instance):
        x_input=TextInput(text='0', multiline=False)
        base_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            x=float(x_input.text)
            base=float(base_input.text)
            result = math.log(x,base)
            self.result_label.text = f'Result: The logarithm of 
{x} to base {base} is {result}. '

        calc_button = Button(text='Calculate', 
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Enter the Number  '))
```

```
        self.layout.add_widget(x_input)
        self.layout.add_widget(Label(text='Enter the Base  '))
        self.layout.add_widget(base_input)
        self.layout.add_widget(calc_button)

class ITrigonometric(Screen):
    def __init__(self, **kwargs):
        super(ITrigonometric, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.add_widget(self.layout)

        # Add buttons for each formula
        self.layout.add_widget(Button(text=' (theta)=Sin^-
1(Input) (degree) ', on_press=self.calculate_isin))
        self.layout.add_widget(Button(text=' (theta)=Cos^-
1(Input) (degree) ', on_press=self.calculate_icos))
        self.layout.add_widget(Button(text=' (theta)=Tan^-
1(Input) (degree) ', on_press=self.calculate_itan))
        self.layout.add_widget(Button(text=' (theta)=Cot^-
1(Input) (degree) ', on_press=self.calculate_icot))
        self.layout.add_widget(Button(text=' (theta)=Sec^-
1(Input) (degree) ', on_press=self.calculate_isec))
        self.layout.add_widget(Button(text=' (theta)=Cosec^-
1(Input) (degree) ', on_press=self.calculate_icosec))

        # Create a label to display the result
        self.result_label = Label(text='Result: ')

        # Add the label to the grid
        self.layout.add_widget(self.result_label)

        # Formula and caluculations

    def calculate_icosec(self,instance):
        value_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            value=float(value_input.text)

            sin_theta = math.asin(1/value)
            x=(sin_theta)*180/math.pi
            self.result_label.text = f'Result: Value of theta {x}
degree '
```

```
        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input the Value  '))
        self.layout.add_widget(value_input)

        self.layout.add_widget(calc_button)

    def calculate_isec(self,instance):
        value_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            value=float(value_input.text)

            cos_theta = math.acos(1/value)
            x=(cos_theta)*180/math.pi
            self.result_label.text = f'Result:  Value of theta
{x} degree '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input the Value  '))
        self.layout.add_widget(value_input)

        self.layout.add_widget(calc_button)

    def calculate_icot(self,instance):
        value_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            value=float(value_input.text)

            tan_theta = math.atan(1/value)
            x=(tan_theta)*180/math.pi
            self.result_label.text = f'Result:Value of theta {x}
degree '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input the Value  '))
```

```
        self.layout.add_widget(value_input)

        self.layout.add_widget(calc_button)

    def calculate_itan(self,instance):
        value_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            value=float(value_input.text)

            tan_theta = math.atan(value)
            x=(tan_theta)*180/math.pi
            self.result_label.text = f'Result:Value of theta {x}
degree '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input the Value  '))
        self.layout.add_widget(value_input)

        self.layout.add_widget(calc_button)

    def calculate_icos(self,instance):
        value_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            value=float(value_input.text)

            cos_theta = math.acos(value)
            x=(cos_theta)*180/math.pi
            self.result_label.text = f'Result:Value of theta {x}
degree '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input the Value  '))
        self.layout.add_widget(value_input)

        self.layout.add_widget(calc_button)

    def calculate_isin(self,instance):
```

```
        value_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            value=float(value_input.text)

            sin_theta = math.asin(value)
            x=(sin_theta)*180/math.pi
            self.result_label.text = f'Result:Value of theta {x}
degree  '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input the Value  '))
        self.layout.add_widget(value_input)

        self.layout.add_widget(calc_button)
class Trigonometric(Screen):
    def __init__(self, **kwargs):
        super(Trigonometric, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.add_widget(self.layout)

        # Add buttons for each formula
        self.layout.add_widget(Button(text=' Sin(theta) (degree)
', on_press=self.calculate_sin))
        self.layout.add_widget(Button(text=' Cos(theta) (degree)
', on_press=self.calculate_cos))
        self.layout.add_widget(Button(text=' Tan(theta) (degree)
', on_press=self.calculate_tan))
        self.layout.add_widget(Button(text=' Cot(theta) (degree)
', on_press=self.calculate_cot))
        self.layout.add_widget(Button(text=' Sec(theta) (degree)
', on_press=self.calculate_sec))
        self.layout.add_widget(Button(text=' Cosec(theta)
(degree) ', on_press=self.calculate_cosec)
        # Create a label to display the result
        self.result_label = Label(text='Result: ')

        # Add the label to the grid
        self.layout.add_widget(self.result_label)

        # Formula and caluculations
```

```
    def calculate_cosec(self,instance):
        theta_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            theta=float(theta_input.text)
            theta_radians = theta * math.pi / 180
            sin_theta = math.sin(theta_radians)
            x=1/(sin_theta)
            self.result_label.text = f'Result: Value of
Cosec(theta) is {x} degree '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input Value of Theta
(degree) '))
        self.layout.add_widget(theta_input)

        self.layout.add_widget(calc_button)

    def calculate_sec(self,instance):
        theta_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            theta=float(theta_input.text)
            theta_radians = theta * math.pi / 180
            cos_theta = math.cos(theta_radians)
            x=1/(cos_theta)
            self.result_label.text = f'Result: Value of
Sec(theta) is {x} degree '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input Value of Theta
(degree) '))
        self.layout.add_widget(theta_input)

        self.layout.add_widget(calc_button)

    def calculate_cot(self,instance):
        theta_input=TextInput(text='0', multiline=False)
```

```
        def calculate(instance):
            theta=float(theta_input.text)
            theta_radians = theta * math.pi / 180
            tan_theta = math.tan(theta_radians)
            x=1/(tan_theta)
            self.result_label.text = f'Result: Value of
Cot(theta) is {x} degree '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input Value of Theta
(degree) '))
        self.layout.add_widget(theta_input)

        self.layout.add_widget(calc_button)

    def calculate_tan(self,instance):
        theta_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            theta=float(theta_input.text)
            theta_radians = theta * math.pi / 180
            tan_theta = math.tan(theta_radians)
            x=(tan_theta)
            self.result_label.text = f'Result: Value of
Tan(theta) is {x} degree '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input Value of Theta
(degree) '))
        self.layout.add_widget(theta_input)

        self.layout.add_widget(calc_button)

    def calculate_cos(self,instance):
        theta_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            theta=float(theta_input.text)
```

```
                theta_radians = theta * math.pi / 180
                cos_theta = math.cos(theta_radians)
                x=(cos_theta)
                self.result_label.text = f'Result: Value of
Cos(theta) is {x} degree '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input Value of Theta
(degree) '))
        self.layout.add_widget(theta_input)

        self.layout.add_widget(calc_button)

    def calculate_sin(self,instance):
        theta_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            theta=float(theta_input.text)
            theta_radians = theta * math.pi / 180
            sin_theta = math.sin(theta_radians)
            x=(sin_theta)
            self.result_label.text = f'Result: Value of
Sin(theta) is {x} degree '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input Value of Theta
(degree) '))
        self.layout.add_widget(theta_input)

        self.layout.add_widget(calc_button)



class Expansions(Screen):
    def __init__(self, **kwargs):
        super(Expansions, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.add_widget(self.layout)
```

```
        # Add buttons for each formula
         self.layout.add_widget(Button(text='a^2-b^2=(a+b)(a-b)',
on_press=self.calculate_a))
         self.layout.add_widget(Button(text='a^2+b^2=(a+b)^2 -
2ab', on_press=self.calculate_d))
         self.layout.add_widget(Button(text='(a+b)^2= a^2 + 2ab +
b^2', on_press=self.calculate_b))
         self.layout.add_widget(Button(text='(a-b)^2= a^2 - 2ab +
b^2', on_press=self.calculate_c))
         self.layout.add_widget(Button(text='(a+b)^3= a^3 +
3ab(a+b) + b^3', on_press=self.calculate_g))
         self.layout.add_widget(Button(text='(a+b+c)^2= a^2 + b^2
+ c^2 + 2ab + 2bc + 2ca', on_press=self.calculate_e))
         self.layout.add_widget(Button(text='(a-b-c)^2= a^2 + b^2
+ c^2 - 2ab + 2bc - 2ca', on_press=self.calculate_f))
         # Create a label to display the result
         self.result_label = Label(text='Result: ')

         # Add the label to the grid
         self.layout.add_widget(self.result_label)

         # Formula and caluculations

     def calculate_g(self,instance):
         a_input=TextInput(text='0', multiline=False)
         b_input=TextInput(text='0', multiline=False)

         def calculate(instance):
             a=float(a_input.text)
             b=float(b_input.text)
             A=(a**3) + ((3*a*b)*(a+b)) + (b**3)
             self.result_label.text = f'Result: Expansion Solution
is {A} '

         calc_button = Button(text='Calculate',
on_press=calculate)
         # Add the input field and button to the grid
         self.layout.add_widget(Label(text='Input First Number (a)
'))
         self.layout.add_widget(a_input)
         self.layout.add_widget(Label(text='Input Second Number
(b)'))
         self.layout.add_widget(b_input)
```

```
        self.layout.add_widget(calc_button)

    def calculate_f(self,instance):
        a_input=TextInput(text='0', multiline=False)
        b_input=TextInput(text='0', multiline=False)
        c_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            a=float(a_input.text)
            b=float(b_input.text)
            c=float(c_input.text)
            A= (a**2) + (b**2) + (c**2) - (2*a*b) + (2*b*c) -
(2*c*a)
            self.result_label.text = f'Result: Expansion Solution
is {A} '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input First Number (a)
'))
        self.layout.add_widget(a_input)
        self.layout.add_widget(Label(text='Input Second Number
(b)'))
        self.layout.add_widget(b_input)
        self.layout.add_widget(Label(text='Input Third Number
(c)'))
        self.layout.add_widget(c_input)
        self.layout.add_widget(calc_button)


    def calculate_e(self,instance):
        a_input=TextInput(text='0', multiline=False)
        b_input=TextInput(text='0', multiline=False)
        c_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            a=float(a_input.text)
            b=float(b_input.text)
            c=float(c_input.text)
            A= (a**2) + (b**2) + (c**2) + (2*a*b) + (2*b*c) +
(2*c*a)
```

```
            self.result_label.text = f'Result: Expansion Solution
is {A} '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input First Number (a)
'))
        self.layout.add_widget(a_input)
        self.layout.add_widget(Label(text='Input Second Number
(b)'))
        self.layout.add_widget(b_input)
        self.layout.add_widget(Label(text='Input Third Number
(c)'))
        self.layout.add_widget(c_input)
        self.layout.add_widget(calc_button)

    def calculate_d(self,instance):
        a_input=TextInput(text='0', multiline=False)
        b_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            a=float(a_input.text)
            b=float(b_input.text)
            A=((a+b)**2)-(2*a*b)
            self.result_label.text = f'Result: Expansion Solution
is {A} '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input First Number (a)
'))
        self.layout.add_widget(a_input)
        self.layout.add_widget(Label(text='Input Second Number
(b)'))
        self.layout.add_widget(b_input)
        self.layout.add_widget(calc_button)

    def calculate_c(self,instance):
        a_input=TextInput(text='0', multiline=False)
        b_input=TextInput(text='0', multiline=False)
```

```
        def calculate(instance):
            a=float(a_input.text)
            b=float(b_input.text)
            A=(a**2) - (2*a*b) + (b**2)
            self.result_label.text = f'Result: Expansion Solution
is {A} '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input First Number (a)
'))
        self.layout.add_widget(a_input)
        self.layout.add_widget(Label(text='Input Second Number
(b)'))
        self.layout.add_widget(b_input)
        self.layout.add_widget(calc_button)

    def calculate_b(self,instance):
        a_input=TextInput(text='0', multiline=False)
        b_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            a=float(a_input.text)
            b=float(b_input.text)
            A=(a**2) + (2*a*b) + (b**2)
            self.result_label.text = f'Result: Expansion Solution
is {A} '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input First Number (a)
'))
        self.layout.add_widget(a_input)
        self.layout.add_widget(Label(text='Input Second Number
(b)'))
        self.layout.add_widget(b_input)
        self.layout.add_widget(calc_button)

    def calculate_a(self,instance):
        a_input=TextInput(text='0', multiline=False)
        b_input=TextInput(text='0', multiline=False)
```

```
        def calculate(instance):
            a=float(a_input.text)
            b=float(b_input.text)
            A=(a+b)*(a-b)
            self.result_label.text = f'Result: Expansion Solution
is {A} '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input First Number (a)
'))
        self.layout.add_widget(a_input)
        self.layout.add_widget(Label(text='Input Second Number
(b)'))
        self.layout.add_widget(b_input)
        self.layout.add_widget(calc_button)



class Perimeter(Screen):
    def __init__(self, **kwargs):
        super(Perimeter, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.add_widget(self.layout)

        # Add buttons for each formula

        self.layout.add_widget(Button(text='PERIMETER OF SQUARE
P=4*(side)', on_press=self.calculate_square))
        self.layout.add_widget(Button(text='PERIMETER OF
RECTANGLE P=2*(L*B)', on_press=self.calculate_rectangle))
        self.layout.add_widget(Button(text='PERIMETER OF CIRCLE
P=2*(pi)*r ', on_press=self.calculate_circle))
        self.layout.add_widget(Button(text='PERIMETER OF TRIANGLE
P=s1 + s2 + b ', on_press=self.calculate_triangle))
        self.layout.add_widget(Button(text='PERIMETER OF
TRAPEZOID P=h + B1 + B2 + c ', on_press=self.calculate_trape))
        self.layout.add_widget(Button(text='PERIMETER OF RHOMBUS
P=4*(side)', on_press=self.calculate_rhomb))
        self.layout.add_widget(Button(text='PERIMETER OF REGULAR
POLYGON P=n*s', on_press=self.calculate_poly))
```

```
        # Create a label to display the result
        self.result_label = Label(text='Result: ')

        # Add the label to the grid
        self.layout.add_widget(self.result_label)

        # Formula and caluculations

    def calculate_poly(self,instance):
        n_input=TextInput(text='0', multiline=False)
        s_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            n=float(n_input.text)
            s=float(s_input.text)
            a=n*s
            self.result_label.text = f'Result: {a} m'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Number of Sides (n)
'))
        self.layout.add_widget(n_input)
        self.layout.add_widget(Label(text='Length of Side (s)'))
        self.layout.add_widget(s_input)
        self.layout.add_widget(calc_button)

    def calculate_rhomb(self,instance):
        side_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            s = float(side_input.text)
            a = s*4
            self.result_label.text = f'Result: {a} m'

        calc_button = Button(text='Calculate',
on_press=calculate)
        self.layout.add_widget(Label(text='Side in m'))
        self.layout.add_widget(side_input)
        self.layout.add_widget(calc_button)

    def calculate_trape(self,instance):
```

```
        B1_input=TextInput(text='0', multiline=False)
        B2_input=TextInput(text='0', multiline=False)
        h_input=TextInput(text='0', multiline=False)
        c_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            B1=float(B1_input.text)
            B2=float(B2_input.text)
            c=float(B2_input.text)
            h=float(h_input.text)
            a=h+B1+B2+c
            self.result_label.text = f'Result: {a} m '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Length of upper
parallel line (B1) in m'))
        self.layout.add_widget(B1_input)
        self.layout.add_widget(Label(text='Length of Lower
parallel line (B2) in m'))
        self.layout.add_widget(B2_input)
        self.layout.add_widget(Label(text='Length of Non parallel
line (c) in m'))
        self.layout.add_widget(c_input)
        self.layout.add_widget(Label(text='Height (h) in m'))
        self.layout.add_widget(h_input)
        self.layout.add_widget(calc_button)

    def calculate_triangle(self,instance):
        s1_input=TextInput(text='0', multiline=False)
        breadth_input=TextInput(text='0', multiline=False)
        s2_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            s1=float(s1_input.text)
            s2=float(s2_input.text)
            b=float(breadth_input.text)
            a=s2+b+s1
            self.result_label.text = f'Result: {a} m'

        calc_button = Button(text='Calculate',
on_press=calculate)
```

```
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Side First (s1) in
m'))
        self.layout.add_widget(s1_input)
        self.layout.add_widget(Label(text='Side Second (s2) in
m'))
        self.layout.add_widget(s2_input)
        self.layout.add_widget(Label(text='Base (b) in m'))
        self.layout.add_widget(breadth_input)
        self.layout.add_widget(calc_button)

    def calculate_circle(self,instance):
        radius_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            r=float(radius_input.text)
            a=2*3.14*r
            self.result_label.text = f'Result: {a} m'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Radius in m'))
        self.layout.add_widget(radius_input)

        self.layout.add_widget(calc_button)

    def calculate_rectangle(self,instance):
        length_input=TextInput(text='0', multiline=False)
        breadth_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            l=float(length_input.text)
            b=float(breadth_input.text)
            a=2*l*b
            self.result_label.text = f'Result: {a} m'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Length in m'))
        self.layout.add_widget(length_input)
        self.layout.add_widget(Label(text='Breadth in m'))
```

```
        self.layout.add_widget(breadth_input)
        self.layout.add_widget(calc_button)

    def calculate_square(self,instance):
        side_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            s = float(side_input.text)
            a = s*4
            self.result_label.text = f'Result: {a} m'

        calc_button = Button(text='Calculate',
on_press=calculate)
        self.layout.add_widget(Label(text='Side in m'))
        self.layout.add_widget(side_input)
        self.layout.add_widget(calc_button)

class Area(Screen):
    def __init__(self, **kwargs):
        super(Area, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.add_widget(self.layout)

        # Add buttons for each formula

        self.layout.add_widget(Button(text='AREA OF SQUARE
A=(Side)^2', on_press=self.calculate_square))
        self.layout.add_widget(Button(text='AREA OF CIRCLE
A=(pi)r^2', on_press=self.calculate_circle))
        self.layout.add_widget(Button(text='AREA OF RECTANGLE
A=L*W', on_press=self.calculate_rectangle))
        self.layout.add_widget(Button(text='AREA OF TRIANGLE
A=(1/2)B*h', on_press=self.calculate_triangle))
        self.layout.add_widget(Button(text='AREA OF TRAPEZOIDAL
A=(1/2)*(B1+B2)*h', on_press=self.calculate_trapezoidal))
        self.layout.add_widget(Button(text='AREA OF PARALLELOGRAM
A=b*h', on_press=self.calculate_parall))
        self.layout.add_widget(Button(text='AREA OF RHOMBUS
A=(1/2)d1*d2', on_press=self.calculate_rhomb))
        self.layout.add_widget(Button(text='AREA OF REGULAR
POLYGON A=(1/2)P*A', on_press=self.calculate_polygon))

        # Create a label to display the result
```

```
        self.result_label = Label(text='Result: ')

        # Add the label to the grid
        self.layout.add_widget(self.result_label)

        # Formula and caluculations

    def calculate_polygon(self,instance):
        d1_input=TextInput(text='0', multiline=False)
        d2_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            d1=float(d1_input.text)
            d2=float(d2_input.text)
            a=(d1*d2)/2
            self.result_label.text = f'Result: {a} m^2'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Length of an Apothem
(A) '))
        self.layout.add_widget(d1_input)
        self.layout.add_widget(Label(text='Perimeter (P)'))
        self.layout.add_widget(d2_input)
        self.layout.add_widget(calc_button)

    def calculate_rhomb(self,instance):
        d1_input=TextInput(text='0', multiline=False)
        d2_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            d1=float(d1_input.text)
            d2=float(d2_input.text)
            a=(d1*d2)/2
            self.result_label.text = f'Result: {a} m^2'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Length of 1st Diagonal
in m'))
        self.layout.add_widget(d1_input)
```

```
        self.layout.add_widget(Label(text='Length of 2nd Diagonal
in m'))
        self.layout.add_widget(d2_input)
        self.layout.add_widget(calc_button)

    def calculate_parall(self,instance):
        height_input=TextInput(text='0', multiline=False)
        base_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            h=float(height_input.text)
            b=float(base_input.text)
            a=h*b
            self.result_label.text = f'Result: {a} m^2'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Height (h) in m'))
        self.layout.add_widget(height_input)
        self.layout.add_widget(Label(text='Base (b) in m'))
        self.layout.add_widget(base_input)
        self.layout.add_widget(calc_button)

    def calculate_trapezoidal(self,instance):
        B1_input=TextInput(text='0', multiline=False)
        B2_input=TextInput(text='0', multiline=False)
        h_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            B1=float(B1_input.text)
            B2=float(B2_input.text)
            h=float(h_input.text)
            a=((B1+B2)/2)*h
            self.result_label.text = f'Result: {a} m^2'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Length of upper
parallel line (B1) in m'))
        self.layout.add_widget(B1_input)
```

```
        self.layout.add_widget(Label(text='Length of Lower
parallel line (B2) in m'))
        self.layout.add_widget(B2_input)
        self.layout.add_widget(Label(text='Height (h) in m'))
        self.layout.add_widget(h_input)
        self.layout.add_widget(calc_button)

    def calculate_triangle(self,instance):
        length_input=TextInput(text='0', multiline=False)
        breadth_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            l=float(length_input.text)
            b=float(breadth_input.text)
            a=(1/2)*l*b
            self.result_label.text = f'Result: {a} m^2'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Length in m'))
        self.layout.add_widget(length_input)
        self.layout.add_widget(Label(text='Base in m'))
        self.layout.add_widget(breadth_input)
        self.layout.add_widget(calc_button)

    def calculate_rectangle(self,instance):
        length_input=TextInput(text='0', multiline=False)
        breadth_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            l=float(length_input.text)
            b=float(breadth_input.text)
            a=l*b
            self.result_label.text = f'Result: {a} m^2'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Length in m'))
        self.layout.add_widget(length_input)
        self.layout.add_widget(Label(text='Breadth in m'))
        self.layout.add_widget(breadth_input)
```

```
        self.layout.add_widget(calc_button)


    def calculate_circle(self,instance):
        radius_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            r=float(radius_input.text)
            a=3.14*r*r
            self.result_label.text = f'Result: {a} m^2'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Radius in m'))
        self.layout.add_widget(radius_input)

        self.layout.add_widget(calc_button)

    def calculate_square(self,instance):
        side_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            s = float(side_input.text)
            a = s ** 2
            self.result_label.text = f'Result: {a} m^2'

        calc_button = Button(text='Calculate',
on_press=calculate)
        self.layout.add_widget(Label(text='Side in m'))
        self.layout.add_widget(side_input)
        self.layout.add_widget(calc_button)
class PhysicsCalc(Screen):
    def __init__(self, **kwargs):
        super(PhysicsCalc, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.classical_button = Button(text='CLASSICAL
MECHANICS', font_size=24, size_hint=(1, 0.3), bold=True,
color=[1, 1, 1, 1],
                                      background_color=[0.1, 0.4,
0.8, 1], on_press=self.classical_press)
        self.thermo_button = Button(text='THERMODYNAMICS',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
```

```
                                                 background_color=[0.1, 0.4,
0.8, 1], on_press=self.thermo_press)
        self.electromag_button = Button(text='ELECTROMAGNETISM',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                                 background_color=[0.1, 0.4,
0.8, 1], on_press=self.electromag_press)
        self.optics_button = Button(text='OPTICS', font_size=24,
size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                                 background_color=[0.1, 0.4,
0.8, 1], on_press=self.optics_press)
        self.elasticity_button = Button(text='ELASTICITY',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                                 background_color=[0.1, 0.4,
0.8, 1], on_press=self.elasticity_press)
        self.astro_button = Button(text='ASTROPHYSICS',
font_size=24, size_hint=(1, 0.3), bold=True, color=[1, 1, 1, 1],
                                                 background_color=[0.1, 0.4,
0.8, 1], on_press=self.astro_press)


        self.layout.add_widget(self.classical_button)
        self.layout.add_widget(self.thermo_button)
        self.layout.add_widget(self.electromag_button)
        self.layout.add_widget(self.optics_button)
        self.layout.add_widget(self.elasticity_button)
        self.layout.add_widget(self.astro_button)
        self.add_widget(self.layout)

    def classical_press(self, instance):
        self.manager.current = 'classical'
    def thermo_press(self, instance):
        self.manager.current = 'thermo'
    def electromag_press(self, instance):
        self.manager.current = 'electromag'
    def optics_press(self, instance):
        self.manager.current = 'optics'
    def elasticity_press(self, instance):
        self.manager.current = 'elast'
    def astro_press(self, instance):
        self.manager.current = 'astro'

class Astrophysics(Screen):
    def __init__(self, **kwargs):
```

```
        super(Astrophysics, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.add_widget(self.layout)

        # Add buttons for each formula

        self.layout.add_widget(Button(text='ENERGY E = mc^2',
on_press=self.calculate_energy))
        self.layout.add_widget(Button(text='GRAVITATIONAL FORCE
F=GMm/r^2', on_press=self.calculate_grav))
        self.layout.add_widget(Button(text='STEFAN BOLTZMAN LAW
L=(sigma)*T^4', on_press=self.calculate_stfn))
        self.layout.add_widget(Button(text='SCHWARZSCHILD RADIUS
Rs=2Gm/c^2', on_press=self.calculate_schwar))
        self.layout.add_widget(Button(text='HUBBLES LAW V=Ho*D',
on_press=self.calculate_HoD))

        # Create a label to display the result
        self.result_label = Label(text='Result: ')

        # Add the label to the grid
        self.layout.add_widget(self.result_label)

        # Formula and caluculations

    def calculate_HoD(self, instance):
        distance_input = TextInput(text='0', multiline=False)

        def calculate(instance):
            D = float(distance_input.text)
            H = 69.8  # speed of light in meters per second
            V = H*D
            self.result_label.text = f'Result: Recessional
Velocity is {V} km/s'

        # Create a "Calculate" button that calls the callback
function when pressed
        calc_button = Button(text='Calculate',
on_press=calculate)

        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Proper Distance D (km)
:'))
```

```
        self.layout.add_widget(distance_input)
        self.layout.add_widget(calc_button)

    def calculate_schwar(self, instance):

        mass_input = TextInput(text='0', multiline=False)


        def calculate(instance):
            m = float(mass_input.text)
            G = (6.67430*(10**(-11)))
            c = 299792458
            R=(2*G*m)/(c**2)
            self.result_label.text = f'Result: Event Horizon of a
Schwarzchild Black Hole (Rs) is{R} '

        calc_button = Button(text='Calculate',
on_press=calculate)

        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Mass (kg) :'))
        self.layout.add_widget(mass_input)
        self.layout.add_widget(calc_button)

    def calculate_stfn(self, instance):

        temp_input = TextInput(text='0', multiline=False)

        def calculate(instance):
            t = float(temp_input.text)
            s = (5.6703*(10**(-8)))
            L = s*(t**4)
            self.result_label.text = f'Result: The Black body
radiant Emittance is {L} J'

        # Create a "Calculate" button that calls the callback
function when pressed
        calc_button = Button(text='Calculate',
on_press=calculate)

        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input Thermodynamic
Temperature (kelvin) :'))
```

```
        self.layout.add_widget(temp_input)
        self.layout.add_widget(calc_button)


    def calculate_energy(self, instance):
        # Create input field for mass
        mass_input = TextInput(text='0', multiline=False)

        # Create a callback function that calculates the energy
when the user presses the "Calculate" button
        def calculate(instance):
            m = float(mass_input.text)
            c = 299792458  # speed of light in meters per second
            E = m * c**2
            self.result_label.text = f'Result: {E} J'

        # Create a "Calculate" button that calls the callback
function when pressed
        calc_button = Button(text='Calculate',
on_press=calculate)

        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Mass (kg):'))
        self.layout.add_widget(mass_input)
        self.layout.add_widget(calc_button)

    def calculate_grav(self,instance):
        Mass_input=TextInput(text='0', multiline=False)
        mass_input=TextInput(text='0', multiline=False)
        r_input=TextInput(text='0', multiline=False)
        def calculate(instance):
            G=6.67*(10**(-11))
            M=float(Mass_input.text)
            m=float(mass_input.text)
            r=float(r_input.text)
            F=(G*M*m)/r**2
            self.result_label.text = f'Result: {F} newton'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Mass of one object
:'))
```

```
        self.layout.add_widget(Mass_input)
        self.layout.add_widget(Label(text='Mass of Second object
:'))
        self.layout.add_widget(mass_input)
        self.layout.add_widget(Label(text='Distance between the
two objects :'))
        self.layout.add_widget(r_input)
        self.layout.add_widget(calc_button)



class Elasticity(Screen):
    def __init__(self, **kwargs):
        super(Elasticity, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.add_widget(self.layout)

        # Add buttons for each formula

        self.layout.add_widget(Button(text='STRESS [S=F/A]',
on_press=self.calculate_stress))
        self.layout.add_widget(Button(text='STRAIN [S=l/L]',
on_press=self.calculate_strain))
        self.layout.add_widget(Button(text='VOLUME STRAIN
[Vs=dV/V]', on_press=self.calculate_Vstrain))
        self.layout.add_widget(Button(text='YOUNGs MODULUS
[E=stress/strain]', on_press=self.calculate_ygmd))
        self.layout.add_widget(Button(text='BULK MODULUS
[B=V*(delta_p)/(delta_v)]', on_press=self.calculate_blkmd))

        # Create a label to display the result
        self.result_label = Label(text='Result: ')

        # Add the label to the grid
        self.layout.add_widget(self.result_label)

        # Formula and caluculations

    def calculate_blkmd(self,instance):
        stress_input=TextInput(text='0', multiline=False)
        strain_input=TextInput(text='0', multiline=False)
        volm_input=TextInput(text='0', multiline=False)
```

```
        def calculate(instance):
            a=float(stress_input.text)
            b=float(strain_input.text)
            v=float(volm_input.text)
            S=-(a*v)/b
            self.result_label.text = f'Result: The Bulk Modulus
is {S} N/m^2 '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Change in pressure
(delta_p) pascal :'))
        self.layout.add_widget(stress_input)
        self.layout.add_widget(Label(text='Change in Volume
(delta_v) m^3 :'))
        self.layout.add_widget(strain_input)
        self.layout.add_widget(Label(text='Volume (m^3) :'))
        self.layout.add_widget(volm_input)
        self.layout.add_widget(calc_button)

    def calculate_ygmd(self,instance):
        stress_input=TextInput(text='0', multiline=False)
        strain_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            a=float(stress_input.text)
            b=float(strain_input.text)
            S=a/b
            self.result_label.text = f'Result: The Youngs Modulus
is {S} N/m^2 '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Input the Stress :'))
        self.layout.add_widget(stress_input)
        self.layout.add_widget(Label(text='Input the Strain :'))
        self.layout.add_widget(strain_input)
        self.layout.add_widget(calc_button)

    def calculate_Vstrain(self,instance):
        l_input=TextInput(text='0', multiline=False)
```

```
        L_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            l=float(l_input.text)
            L=float(L_input.text)
            S=l/L
            self.result_label.text = f'Result: The Strain is {S}
'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Change in Volume (dV)
:'))
        self.layout.add_widget(l_input)
        self.layout.add_widget(Label(text='Original Volume (v)
:'))
        self.layout.add_widget(L_input)
        self.layout.add_widget(calc_button)

    def calculate_strain(self,instance):
        l_input=TextInput(text='0', multiline=False)
        L_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            l=float(l_input.text)
            L=float(L_input.text)
            S=l/L
            self.result_label.text = f'Result: The Strain is {S}
'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Change in length (l) m
:'))
        self.layout.add_widget(l_input)
        self.layout.add_widget(Label(text='Original length (L) m
:'))
        self.layout.add_widget(L_input)
        self.layout.add_widget(calc_button)

    def calculate_stress(self,instance):
```

```
        force_input=TextInput(text='0', multiline=False)
        area_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            f=float(force_input.text)
            a=float(area_input.text)
            S=f/a
            self.result_label.text = f'Result: The Stress is {S}
pascal'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Applied Force (F) N
:'))
        self.layout.add_widget(force_input)
        self.layout.add_widget(Label(text='Area (A) m^2:'))
        self.layout.add_widget(area_input)
        self.layout.add_widget(calc_button)

class opticss(Screen):
    def __init__(self, **kwargs):
        super(opticss, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.add_widget(self.layout)

        # Add buttons for each formula

        self.layout.add_widget(Button(text='DIFFRACTION GRATING
[n*(lambda)=sin(Theta)/N]', on_press=self.calculate_grating))
        self.layout.add_widget(Button(text='BEER-LAMBERT LAW
[A=(Epsilon)*b*c]', on_press=self.calculate_beer))
        self.layout.add_widget(Button(text='SNELLs LAW (Incident
angle) theta=sin^-1(sin(t)n2/n1)]',
on_press=self.calculate_snell1))
        self.layout.add_widget(Button(text='SNELLs LAW (Refracted
angle) theta=sin^-1(sin(t)n1/n2)]',
on_press=self.calculate_snell2))
        self.layout.add_widget(Button(text='CRITICAL ANGLE
[C=sin^-1(Nr/Ni)]', on_press=self.calculate_critical))
        self.layout.add_widget(Button(text='RAYLEIGHs CRITERIAN
[(Theta)=1.22*(Lambda)/D]', on_press=self.calculate_rayleigh))
        # Create a label to display the result
```

```
        self.result_label = Label(text='Result: ')

        # Add the label to the grid
        self.layout.add_widget(self.result_label)

        # Formula and caluculations

    def calculate_rayleigh(self,instance):
        l_input=TextInput(text='0', multiline=False)
        d_input=TextInput(text='0', multiline=False)


        def calculate(instance):
            l=float(l_input.text)
            d=float(d_input.text)
            t=(1.22*l)/d
            self.result_label.text = f'Result: Angle (Theta) is
{t} '

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Wavelength (Lambda)
:'))
        self.layout.add_widget(l_input)
        self.layout.add_widget(Label(text='Diameter of the beam
(D):'))
        self.layout.add_widget(d_input)
        self.layout.add_widget(calc_button)

    def calculate_critical(self,instance):
        n1_input=TextInput(text='0', multiline=False)
        n2_input=TextInput(text='0', multiline=False)
        def calculate(instance):
            a=float(n1_input.text)
            b=float(n2_input.text)
            x=a/b
            y= math.asin(x)
            t2=(y*180)/math.pi
            self.result_label.text = f'Result:Critical Angle is
{t2} degree'
```

```
        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Refracted Index (Nr)
:'))
        self.layout.add_widget(n1_input)
        self.layout.add_widget(Label(text='Incident Index (Ni)
:'))
        self.layout.add_widget(n2_input)
        self.layout.add_widget(calc_button)

    def calculate_snell2(self,instance):
        n1_input=TextInput(text='0', multiline=False)
        n2_input=TextInput(text='0', multiline=False)
        theta_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            n1=float(n1_input.text)
            n2=float(n2_input.text)
            t1=float(theta_input.text)
            theta_radians = t1 * math.pi / 180
            sin_theta = math.sin(theta_radians)
            x=(sin_theta)*(n1/n2)
            y=math.asin(x)
            t2=(y*180)/math.pi
            self.result_label.text = f'Result:Using Snells law
Refracted angle is {t2} degree'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Incident Index (n1)
:'))
        self.layout.add_widget(n1_input)
        self.layout.add_widget(Label(text='Refracted Index (n2)
:'))
        self.layout.add_widget(n2_input)
        self.layout.add_widget(Label(text='Incident Angle
(Degree):'))
        self.layout.add_widget(theta_input)
        self.layout.add_widget(calc_button)

    def calculate_snell1(self,instance):
```

```
        n1_input=TextInput(text='0', multiline=False)
        n2_input=TextInput(text='0', multiline=False)
        theta_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            n1=float(n1_input.text)
            n2=float(n2_input.text)
            t2=float(theta_input.text)
            theta_radians = t2 * math.pi / 180
            sin_theta = math.sin(theta_radians)
            x=(sin_theta)*(n2/n1)
            y=math.asin(x)
            t1=(y*180)/math.pi
            self.result_label.text = f'Result:Using Snells law
Incident angle is {t1} degree'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Incident Index (n1)
:'))
        self.layout.add_widget(n1_input)
        self.layout.add_widget(Label(text='Refracted Index (n2)
:'))
        self.layout.add_widget(n2_input)
        self.layout.add_widget(Label(text='Refracted Angle
(Degree):'))
        self.layout.add_widget(theta_input)
        self.layout.add_widget(calc_button)


    def calculate_beer(self,instance):
        e_input=TextInput(text='0', multiline=False)
        b_input=TextInput(text='0', multiline=False)
        c_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            e=float(e_input.text)
            b=float(b_input.text)
            c=float(c_input.text)
            A=e*b*c
            self.result_label.text = f'Result: The Absorbance {A}
AU'
```

```
        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Molar Absorptivity
(Epsilon) :'))
        self.layout.add_widget(e_input)
        self.layout.add_widget(Label(text='Length of light path
(b):'))
        self.layout.add_widget(b_input)
        self.layout.add_widget(Label(text='Concentration (c):'))
        self.layout.add_widget(c_input)
        self.layout.add_widget(calc_button)

    def calculate_grating(self,instance):
        n_input=TextInput(text='0', multiline=False)
        N_input=TextInput(text='0', multiline=False)
        theta_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            n=float(n_input.text)
            N=float(N_input.text)
            t=float(theta_input.text)
            theta_radians = t * math.pi / 180
            sin_theta = math.sin(theta_radians)
            w=(sin_theta)/(n*N)
            self.result_label.text = f'Result: The Wavelength
will be {w} meter'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Integral Multiple of
Wavelength (n) :'))
        self.layout.add_widget(n_input)
        self.layout.add_widget(Label(text='Number of Lines per
unit Length (N) :'))
        self.layout.add_widget(N_input)
        self.layout.add_widget(Label(text='Angle of emergence
(Theta) :'))
        self.layout.add_widget(theta_input)
        self.layout.add_widget(calc_button)
class Electromagnetism(Screen):
```

```
    def __init__(self, **kwargs):
        super(Electromagnetism, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.add_widget(self.layout)

        # Add buttons for each formula
        self.layout.add_widget(Button(text='COULOMBS LAW
F=k(q1q2/r^2)', on_press=self.calculate_coulomb))
        self.layout.add_widget(Button(text='ELECTRIC POTENTIAL
V=k(q/r)', on_press=self.calculate_elept))
        self.layout.add_widget(Button(text='ELECTRIC FIELD
E=k|Q|/r^2 ', on_press=self.calculate_eleFeild))
        self.layout.add_widget(Button(text='GAUSS LAW Phi =
Q/Epsilon_0', on_press=self.calculate_gauss))
        self.layout.add_widget(Button(text='MAGNETIC FLUX DENSITY
B=(4pi*10^-7*I)/(2pi*R)', on_press=self.calculate_amp))
        self.layout.add_widget(Button(text='FARADAYS LAW
(Epsilon)=-N(*Phi)/(*t)', on_press=self.calculate_fard))


        # Create a label to display the result
        self.result_label = Label(text='Result: ')

        # Add the label to the grid
        self.layout.add_widget(self.result_label)

        # Formula and caluculations


    def calculate_fard(self,instance):
        n_input=TextInput(text='0', multiline=False)
        m_input=TextInput(text='0', multiline=False)
        t_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            n=float(n_input.text)
            m=float(m_input.text)
            t=float(t_input.text)
            F=-((n*m)/t)
            self.result_label.text = f'Result: Induced Voltage
(Epsilon) is {F} weber'
```

```
        calc_button = Button(text='Calculate', 
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text=' Number of Loop :'))
        self.layout.add_widget(n_input)
        self.layout.add_widget(Label(text='Change in Magnetic 
Flux (Delta_phi) (tesla) :'))
        self.layout.add_widget(m_input)
        self.layout.add_widget(Label(text='Change in Time 
(Delta_t) (second) :'))
        self.layout.add_widget(t_input)
        self.layout.add_widget(calc_button)

    def calculate_amp(self,instance):
        current_input=TextInput(text='0', multiline=False)
        radius_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            i=float(current_input.text)
            r=float(radius_input.text)
            B=2*(10**(-7))*(i/r)
            self.result_label.text = f'Result: Magnetic Flux 
Density (B) is {B} tesla'

        calc_button = Button(text='Calculate', 
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text=' Current I (Ampere) 
:'))
        self.layout.add_widget(current_input)
        self.layout.add_widget(Label(text='Length or radius R 
(meter) :'))
        self.layout.add_widget(radius_input)
        self.layout.add_widget(calc_button)


    def calculate_gauss(self,instance):
        q_input=TextInput(text='0', multiline=False)
        a_input=TextInput(text='0', multiline=False)

        def calculate(instance):
            q=float(q_input.text)
            a=float(a_input.text)
```

```
            g=q/a
            self.result_label.text = f'Result: electric flux
(Phi) is {g} Nm^2/C'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text=' Total charge enclosed
within V :'))
        self.layout.add_widget(q_input)
        self.layout.add_widget(Label(text='electric constant
(Epsilon_0) :'))
        self.layout.add_widget(a_input)
        self.layout.add_widget(calc_button)

    def calculate_eleFeild(self,instance):
        q_input=TextInput(text='0', multiline=False)
        r_input=TextInput(text='0', multiline=False)
        def calculate(instance):
            k=8.99*(10**(9))
            r=float(r_input.text)
            q=float(q_input.text)
            E=(k*q)/r**2
            self.result_label.text = f'Result: {E}
newton/coulombs'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Charge |Q| (coulombs)
:'))
        self.layout.add_widget(q_input)
        self.layout.add_widget(Label(text='Distance of seperation
r (meter) :'))
        self.layout.add_widget(r_input)
        self.layout.add_widget(calc_button)

    def calculate_elept(self,instance):
        q_input=TextInput(text='0', multiline=False)
        r_input=TextInput(text='0', multiline=False)
        def calculate(instance):
            k=8.99*(10**(9))
            q=float(q_input.text)
```

```
            r=float(r_input.text)
            V=(k*q)/r
            self.result_label.text = f'Result: {V}
joules/coulombs'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Charge q (coulombs)
:'))
        self.layout.add_widget(q_input)
        self.layout.add_widget(Label(text='Distance of seperation
(meter) :'))
        self.layout.add_widget(r_input)
        self.layout.add_widget(calc_button)

    def calculate_coulomb(self,instance):
        q1_input=TextInput(text='0', multiline=False)
        q2_input=TextInput(text='0', multiline=False)
        r_input=TextInput(text='0', multiline=False)
        def calculate(instance):
            k=8.99*(10**(9))
            q1=float(q1_input.text)
            q2=float(q2_input.text)
            r=float(r_input.text)
            F=(k*q1*q2)/r**2
            self.result_label.text = f'Result: {F} newton'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='First Charge q1 :'))
        self.layout.add_widget(q1_input)
        self.layout.add_widget(Label(text='Second Charge q2 :'))
        self.layout.add_widget(q2_input)
        self.layout.add_widget(Label(text='Distance of seperation
:'))
        self.layout.add_widget(r_input)
        self.layout.add_widget(calc_button)
class Thermodynamics(Screen):
    def __init__(self, **kwargs):
        super(Thermodynamics, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
```

```
        self.add_widget(self.layout)

        # Add buttons for each formula
        self.layout.add_widget(Button(text='SPECIFIC HEAT
CAPACITY C= (Delta Q)/m(Delta T)',
on_press=self.calculate_sHeat))
        self.layout.add_widget(Button(text='HEAT ENERGY CAPACITY
(Delta Q)=m * c * (Delta T)', on_press=self.calculate_hCapacity))
        self.layout.add_widget(Button(text='1st LAW OF
THERMODYNAMICS (Delta U)= Q-W', on_press=self.calculate_law1))
        self.layout.add_widget(Button(text='ENTHALPY (H)= E +
PV', on_press=self.calculate_enthalpy))
        self.layout.add_widget(Button(text='GIBBs FREE ENERGY
(Delta G)=(Delta H)- T(Delta S)', on_press=self.calculate_gibbs))
        self.layout.add_widget(Button(text='HELMHOLTZ FREE ENERGY
F=U-TS', on_press=self.calculate_helms))
        self.layout.add_widget(Button(text='LATENT HEAT L=Q/M',
on_press=self.calculate_latent))



        # Create a label to display the result
        self.result_label = Label(text='Result: ')

        # Add the label to the grid
        self.layout.add_widget(self.result_label)

        # Formula and caluculations

    def calculate_latent(self, instance):
        # Create input field for mass
        Q_input = TextInput(text='0', multiline=False)
        M_input = TextInput(text='0', multiline=False)

       # Create a callback function that calculates the energy
when the user presses the "Calculate" button
        def calculate(instance):
            Q = float(Q_input.text)
            M = float(M_input.text)

            L=Q/M
            self.result_label.text = f'Result: Latent Heat of
substance is {L} joule/kg '
```

```
        # Create a "Calculate" button that calls the callback
function when pressed
        calc_button = Button(text='Calculate',
on_press=calculate)

        # Add the input field and button to the grid

        self.layout.add_widget(Label(text='Heat in joule :'))
        self.layout.add_widget(Q_input)
        self.layout.add_widget(Label(text='Mass in Kg :'))
        self.layout.add_widget(M_input)
        self.layout.add_widget(calc_button)

    def calculate_helms(self, instance):
        # Create input field for mass
        u_input = TextInput(text='0', multiline=False)
        t_input = TextInput(text='0', multiline=False)
        s_input = TextInput(text='0', multiline=False)
       # Create a callback function that calculates the energy
when the user presses the "Calculate" button
        def calculate(instance):
            U = float(u_input.text)
            T = float(t_input.text)
            S = float(s_input.text)
            F=U-(T*S)
            self.result_label.text = f'Result: Helmholtz Free
Energy is {F} joule '

        # Create a "Calculate" button that calls the callback
function when pressed
        calc_button = Button(text='Calculate',
on_press=calculate)

        # Add the input field and button to the grid

        self.layout.add_widget(Label(text='Internal energy of the
system in joule :'))
        self.layout.add_widget(u_input)
        self.layout.add_widget(Label(text='Temperature in kelvin
:'))
        self.layout.add_widget(t_input)
```

```
        self.layout.add_widget(Label(text='Entropy in
j/k.mole:'))
        self.layout.add_widget(s_input)
        self.layout.add_widget(calc_button)

    def calculate_gibbs(self, instance):
        # Create input field for mass
        h_input = TextInput(text='0', multiline=False)
        t_input = TextInput(text='0', multiline=False)
        s_input = TextInput(text='0', multiline=False)
       # Create a callback function that calculates the energy
when the user presses the "Calculate" button
        def calculate(instance):
            H = float(h_input.text)
            T = float(t_input.text)
            S = float(s_input.text)
            G=H-(T*S)
            self.result_label.text = f'Result: Change in Gibbs
free energy is {G} KJ/mole '

        # Create a "Calculate" button that calls the callback
function when pressed
        calc_button = Button(text='Calculate',
on_press=calculate)

        # Add the input field and button to the grid

        self.layout.add_widget(Label(text='Change in enthalpy
(Delta H) in KJ/mole :'))
        self.layout.add_widget(h_input)
        self.layout.add_widget(Label(text='Temperature in kelvin
:'))
        self.layout.add_widget(t_input)
        self.layout.add_widget(Label(text='Change in Entropy
(Delta S) in joule/kelvin.mole :'))
        self.layout.add_widget(s_input)
        self.layout.add_widget(calc_button)


    def calculate_enthalpy(self, instance):
        # Create input field for mass
        energy_input = TextInput(text='0', multiline=False)
        press_input = TextInput(text='0', multiline=False)
```

```
        volm_input = TextInput(text='0', multiline=False)
       # Create a callback function that calculates the energy
when the user presses the "Calculate" button
        def calculate(instance):
            E = float(energy_input.text)
            P = float(press_input.text)
            V = float(volm_input.text)
            H=E+(P*V)
            self.result_label.text = f'Result: Enthalpy is {H}
joule '

        # Create a "Calculate" button that calls the callback
function when pressed
        calc_button = Button(text='Calculate',
on_press=calculate)

        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Internal energy
(joule):'))
        self.layout.add_widget(energy_input)
        self.layout.add_widget(Label(text='Pressure (pascal):'))
        self.layout.add_widget(press_input)
        self.layout.add_widget(Label(text='Volume (m^3):'))
        self.layout.add_widget(volm_input)
        self.layout.add_widget(calc_button)

    def calculate_law1(self, instance):
        # Create input field for mass
        heat_input = TextInput(text='0', multiline=False)
        workd_input = TextInput(text='0', multiline=False)
       # Create a callback function that calculates the energy
when the user presses the "Calculate" button
        def calculate(instance):
            Q = float(heat_input.text)
            W = float(workd_input.text)
            U =Q-W
            self.result_label.text = f'Result: Change in Internal
energy (Delta U) is {U} joule '

        # Create a "Calculate" button that calls the callback
function when pressed
        calc_button = Button(text='Calculate',
on_press=calculate)
```

```
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Heat (joule):'))
        self.layout.add_widget(heat_input)
        self.layout.add_widget(Label(text='Work Done (joule):'))
        self.layout.add_widget(workd_input)
        self.layout.add_widget(calc_button)

    def calculate_hCapacity(self, instance):
        # Create input field for mass
        c_input = TextInput(text='0', multiline=False)
        t_input = TextInput(text='0', multiline=False)
        m_input = TextInput(text='0', multiline=False)

        # Create a callback function that calculates the energy
when the user presses the "Calculate" button
        def calculate(instance):
            c = float(c_input.text)
            t = float(t_input.text)
            m = float(m_input.text)
            q =c*m*t
            self.result_label.text = f'Result: {q} joule '

        # Create a "Calculate" button that calls the callback
function when pressed
        calc_button = Button(text='Calculate',
on_press=calculate)

        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Specific heat capacity
c (joule/kelvin.kg):'))
        self.layout.add_widget(c_input)
        self.layout.add_widget(Label(text='Difference in
Temperature Delta(T) (kelvin):'))
        self.layout.add_widget(t_input)
        self.layout.add_widget(Label(text='Mass (kg):'))
        self.layout.add_widget(m_input)
        self.layout.add_widget(calc_button
    def calculate_sHeat(self, instance):
        # Create input field for mass
        q_input = TextInput(text='0', multiline=False)
        t_input = TextInput(text='0', multiline=False)
        m_input = TextInput(text='0', multiline=False)
```

```
        # Create a callback function that calculates the energy
when the user presses the "Calculate" button
        def calculate(instance):
            q = float(q_input.text)
            t = float(t_input.text)
            m = float(m_input.text)
            c =q/(m*t)
            self.result_label.text = f'Result: {c}
joule/kelvin.kg'

        # Create a "Calculate" button that calls the callback
function when pressed
        calc_button = Button(text='Calculate',
on_press=calculate)

        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Heat capacity Delta(Q)
(joule):'))
        self.layout.add_widget(q_input)
        self.layout.add_widget(Label(text='Difference in
Temperature Delta(T) (kelvin):'))
        self.layout.add_widget(t_input)
        self.layout.add_widget(Label(text='Mass (kg):'))
        self.layout.add_widget(m_input)
        self.layout.add_widget(calc_button)


class classicalMech(Screen):
    def __init__(self, **kwargs):
        super(classicalMech, self).__init__(**kwargs)
        self.layout = BoxLayout(orientation='vertical')
        self.add_widget(self.layout)

        # Add buttons for each formula
        self.layout.add_widget(Button(text='FORCE F = ma',
on_press=self.calculate_force))
        self.layout.add_widget(Button(text='MOMENTUM a=c.v/t',
on_press=self.calculate_momentum))
        self.layout.add_widget(Button(text='VELOCITY v = d/t',
on_press=self.calculate_velocity))
        self.layout.add_widget(Button(text='ACCELERATION
a=c.v/t', on_press=self.calculate_acceleration))
```

```
        self.layout.add_widget(Button(text='WORK W=F*D',
on_press=self.calculate_work))
        self.layout.add_widget(Button(text='ENERGY E = mc^2',
on_press=self.calculate_energy))
        self.layout.add_widget(Button(text='KINETIC ENERGY
K.E=1/2(mv^2)', on_press=self.calculate_ke))
        self.layout.add_widget(Button(text='POTENTIAL ENERGY
P.E=mgh', on_press=self.calculate_pe))
        self.layout.add_widget(Button(text='GRAVITATIONAL FORCE
F=GMm/r^2', on_press=self.calculate_grav))
        self.layout.add_widget(Button(text='TORQUE
T=rFsin(theta)', on_press=self.calculate_torque))
        # Create a label to display the result
        self.result_label = Label(text='Result: ')

        # Add the label to the grid
        self.layout.add_widget(self.result_label)

        # Formula and caluculations

    def calculate_energy(self, instance):
        # Create input field for mass
        mass_input = TextInput(text='0', multiline=False)

        # Create a callback function that calculates the energy
when the user presses the "Calculate" button
        def calculate(instance):
            m = float(mass_input.text)
            c = 299792458  # speed of light in meters per second
            E = m * c**2
            self.result_label.text = f'Result: {E} J'

        # Create a "Calculate" button that calls the callback
function when pressed
        calc_button = Button(text='Calculate',
on_press=calculate)

        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Mass (kg):'))
        self.layout.add_widget(mass_input)
        self.layout.add_widget(calc_button)

    def calculate_torque(self,instance):
```

```
        force_input=TextInput(text='0', multiline=False)
        radius_input=TextInput(text='0', multiline=False)
        angle_input=TextInput(text='0', multiline=False)
        def calculate(instance):
            f=float(force_input.text)
            r=float(radius_input.text)
            a=float(angle_input.text)
            theta_radians = a * math.pi / 180
            sin_theta = math.sin(theta_radians)
            T=f*r*sin_theta
            self.result_label.text = f'Result: {T} newton-meter'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Force (N) :'))
        self.layout.add_widget(force_input)
        self.layout.add_widget(Label(text='Radius (m) :'))
        self.layout.add_widget(radius_input)
        self.layout.add_widget(Label(text='Angle (Theta) between
Force and Radius :'))
        self.layout.add_widget(angle_input)
        self.layout.add_widget(calc_button)

    def calculate_grav(self,instance):
        Mass_input=TextInput(text='0', multiline=False)
        mass_input=TextInput(text='0', multiline=False)
        r_input=TextInput(text='0', multiline=False)
        def calculate(instance):
            G=6.67*(10**(-11))
            M=float(Mass_input.text)
            m=float(mass_input.text)
            r=float(r_input.text)
            F=(G*M*m)/r**2
            self.result_label.text = f'Result: {F} newton'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Mass of one object
:'))
        self.layout.add_widget(Mass_input)
```

```
        self.layout.add_widget(Label(text='Mass of Second object
:'))
        self.layout.add_widget(mass_input)
        self.layout.add_widget(Label(text='Distance between the
two objects :'))
        self.layout.add_widget(r_input)
        self.layout.add_widget(calc_button)

    def calculate_pe(self,instance):
        mass_input=TextInput(text='0', multiline=False)
        height_input=TextInput(text='0', multiline=False)
        def calculate(instance):
            m=float(mass_input.text)
            h=float(height_input.text)
            g= 9.8
            p=m*g*h
            self.result_label.text = f'Result: {p} joules'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Mass(kg):'))
        self.layout.add_widget(mass_input)
        self.layout.add_widget(Label(text='Height(m):'))
        self.layout.add_widget(height_input)
        self.layout.add_widget(calc_button)

    def calculate_ke(self,instance):
        mass_input=TextInput(text='0', multiline=False)
        velocity_input=TextInput(text='0', multiline=False)
        def calculate(instance):
            m=float(mass_input.text)
            v=float(velocity_input.text)
            k=(m*v**2)/2
            self.result_label.text = f'Result: {k} joules'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Mass(kg):'))
        self.layout.add_widget(mass_input)
        self.layout.add_widget(Label(text='Velocity(m/s):'))
        self.layout.add_widget(velocity_input)
```

```
        self.layout.add_widget(calc_button)

    def calculate_work(self,instance):
        force_input=TextInput(text='0', multiline=False)
        dis_input=TextInput(text='0', multiline=False)
        def calculate(instance):
            F=float(force_input.text)
            D=float(dis_input.text)
            W=F*D
            self.result_label.text = f'Result: {W} J'


        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Force(N):'))
        self.layout.add_widget(force_input)
        self.layout.add_widget(Label(text='Displacement(m):'))
        self.layout.add_widget(dis_input)
        self.layout.add_widget(calc_button)


    def calculate_acceleration(self,instance):
        cvelocity_input=TextInput(text='0', multiline=False)
        time_input=TextInput(text='0', multiline=False)
        def calculate(instance):
            cv=float(cvelocity_input.text)
            t=float(time_input.text)
            a=cv/t
            self.result_label.text = f'Result: {a} m/s^2'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Velocity(m/s):'))
        self.layout.add_widget(cvelocity_input)
        self.layout.add_widget(Label(text='Time(s):'))
        self.layout.add_widget(time_input)
        self.layout.add_widget(calc_button)

    def calculate_velocity(self,instance):
        dis_input=TextInput(text='0', multiline=False)
        time_input=TextInput(text='0', multiline=False)
```

```
        def calculate(instance):
            d=float(dis_input.text)
            t=float(time_input.text)
            v=d/t
            self.result_label.text = f'Result: {v} m/s'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Displacement(m):'))
        self.layout.add_widget(dis_input)
        self.layout.add_widget(Label(text='Time(s):'))
        self.layout.add_widget(time_input)
        self.layout.add_widget(calc_button)

    def calculate_momentum(self,instance):
        mass_input=TextInput(text='0', multiline=False)
        velocity_input=TextInput(text='0', multiline=False)
        def calculate(instance):
            m=float(mass_input.text)
            v=float(velocity_input.text)
            mo=m*v
            self.result_label.text = f'Result: {mo} kg.m/s'

        calc_button = Button(text='Calculate',
on_press=calculate)
        # Add the input field and button to the grid
        self.layout.add_widget(Label(text='Mass(m):'))
        self.layout.add_widget(mass_input)
        self.layout.add_widget(Label(text='Velocity(m/s):'))
        self.layout.add_widget(velocity_input)
        self.layout.add_widget(calc_button)

    def calculate_force(self, instance):
        # Create input fields for mass and acceleration
        mass_input = TextInput(text='0', multiline=False)
        accel_input = TextInput(text='0', multiline=False)

        # Create a callback function that calculates the force
when the user presses the "Calculate" button
        def calculate(instance):
            m = float(mass_input.text)
            a = float(accel_input.text)
```

```
            F = m * a
            self.result_label.text = f'Result: {F} N'

       # Create a "Calculate" button that calls the callback
function when pressed
        calc_button = Button(text='Calculate',
on_press=calculate)

        # Add the input fields and button to the grid
        self.layout.add_widget(Label(text='Mass (kg):'))
        self.layout.add_widget(mass_input)
        self.layout.add_widget(Label(text='Acceleration
(m/s^2):'))
        self.layout.add_widget(accel_input)
        self.layout.add_widget(calc_button)

class Calculator(Screen):
    def __init__(self, **kwargs):
        super(Calculator, self).__init__(**kwargs)
        # Create a BoxLayout to hold the UI elements
        layout = BoxLayout(orientation='vertical', spacing=10,
padding=10)

        # Create a label to display the calculator input/output
        self.display = Label(text='0', font_size=50,
halign='right', size_hint=(1, 0.2))
        layout.add_widget(self.display)

        # Create buttons for the calculator
        buttons = [
            ['7', '8', '9', '/'],
            ['4', '5', '6', '*'],
            ['1', '2', '3', '-'],
            ['.', '0', 'C', '+'],
            ['=',]
        ]

        # Add the buttons to the layout
        for row in buttons:
            row_layout = BoxLayout(spacing=10)
            for label in row:
                button = Button(text=label, font_size=30,
size_hint=(0.2, 1))
```

```
                button.bind(on_press=self.on_button_press)
                row_layout.add_widget(button)
            layout.add_widget(row_layout)

        self.add_widget(layout)

    def on_button_press(self, instance):
        # Get the current input/output from the display
        text = self.display.text

        # Handle the different button presses
        if instance.text == 'C':
            # Clear the display
            self.display.text = '0'
        elif instance.text == '=':
            # Evaluate the expression and display the result
            try:
                result = str(eval(text))
                self.display.text = result
            except:
                self.display.text = 'Error'
        else:
            # Append the button label to the display
            if text == '0':
                self.display.text = instance.text
            else:
                self.display.text = text + instance.text

class PhysicsCalcApp(App):
    def build(self):
        sm = ScreenManager()
        sm.add_widget(SplashPage(name='splash'))
        sm.add_widget(MainPage(name='main'))
        sm.add_widget(PhysicsCalc(name='Phy'))
        sm.add_widget(ChemistryCalc(name='chem'))
        sm.add_widget(MathCalc(name='math'))
        sm.add_widget(Calculator(name='Calc'))
        sm.add_widget(classicalMech(name='classical'))
        sm.add_widget(Thermodynamics(name='thermo'))
        sm.add_widget(Electromagnetism(name='electromag'))
        sm.add_widget(opticss(name='optics'))
        sm.add_widget(Elasticity(name='elast'))
        sm.add_widget(Astrophysics(name='astro'))
```

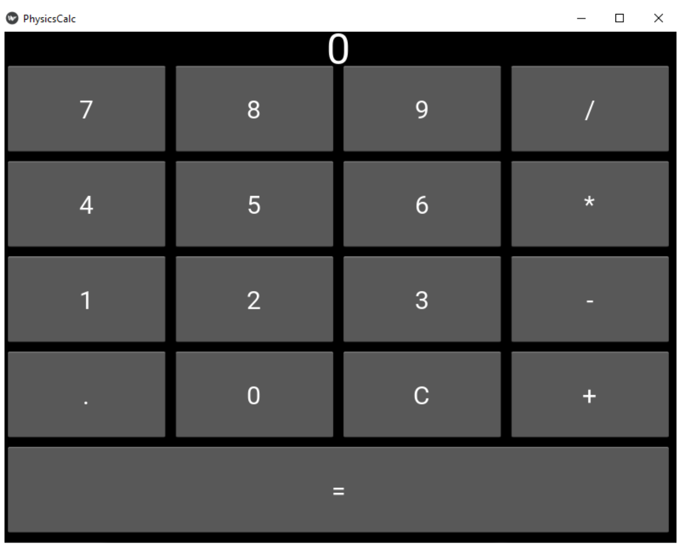
```
        sm.add_widget(MathCalc(name='math'))
        sm.add_widget(Area(name='area'))
        sm.add_widget(Perimeter(name='peri'))
        sm.add_widget(Expansions(name='exp'))
        sm.add_widget(Trigonometric(name='tri'))
        sm.add_widget(ITrigonometric(name='itri'))
        sm.add_widget(Logarithms(name='log'))
        sm.add_widget(Sols(name='sp'))
        return sm

if __name__ == '__main__':
    PhysicsCalcApp().run()
```

CHAPATER 6:

KEY FEATURES AND RESULTS

1. Calculator

The Android application "Science Formula Calculator" includes a key feature, which is a regular calculator. This calculator is used to perform simple arithmetic operations such as addition, subtraction, multiplication, division, and more.



A regular calculator is a basic tool used for performing arithmetic calculations. It typically consists of a numeric keypad with digits from 0 to 9, along with various function keys.

Here is some information about regular calculators:

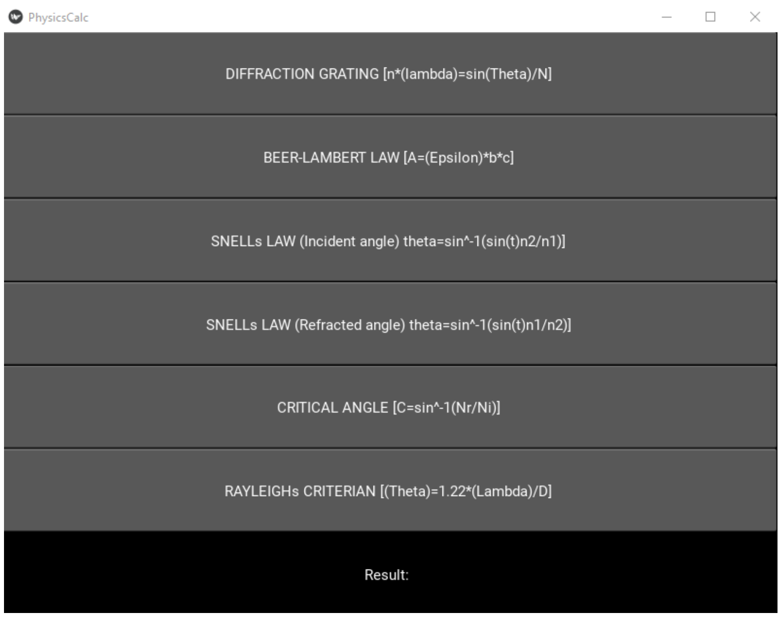
1. Arithmetic Operations: Regular calculators allow users to perform common arithmetic operations, including addition, subtraction, multiplication, and division. They also support the use of parentheses for grouping calculations.

2. Display: Regular calculators feature a digital display that shows the entered numbers and the results of calculations. The display can be either a single-line or multi-line screen, depending on the calculator model.

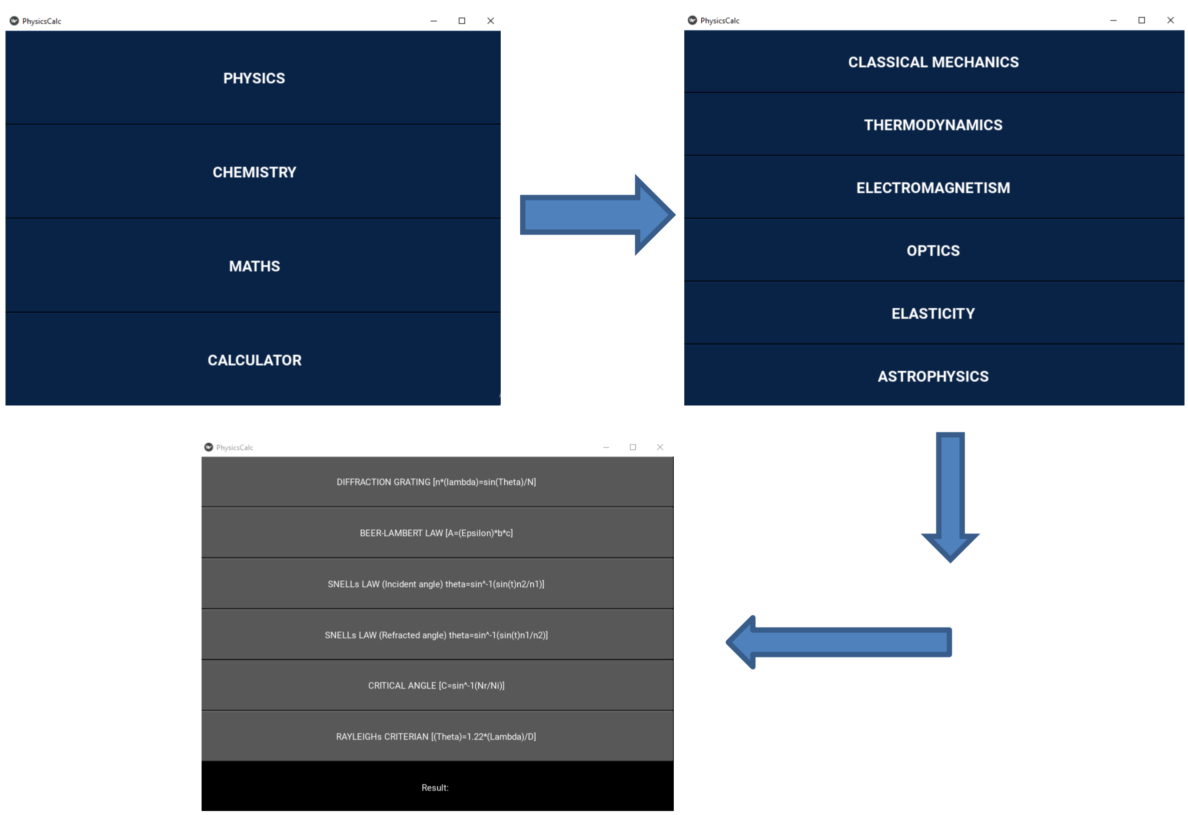
3. Clearing and Correction: Regular calculators usually have a clear key (C/AC) that erases the current calculation or resets the calculator to its initial state. They may also include a backspace or delete key for correcting input errors.

4. Limitations: Regular calculators are designed for basic arithmetic calculations and may not support advanced mathematical functions or complex equations. For more advanced calculations, specialized calculators or software applications are often used.

2. Optics page



How to open optics page



Optics encompasses several complex formulas, some of which involve trigonometric functions. One of the key features of the application is the accurate calculation of formulas such as Snell's law for incident angle and Snell's law for refracted angle. These formulas have been thoroughly cross-checked multiple times to ensure precision and reliability in the calculations

Example

Sample Problem 1:

Light travels from air into an optical fiber with an index of refraction of 1.44. (a) In which direction does the light bend? (b) If the angle of incidence on the end of the fiber is 22°, what is the angle of refraction inside the fiber? (c) Sketch the path of light as it changes media.

Solution:

(a) Since the light is traveling from a rarer region (lower *n*) to a denser region (higher *n*), it will bend toward the normal.

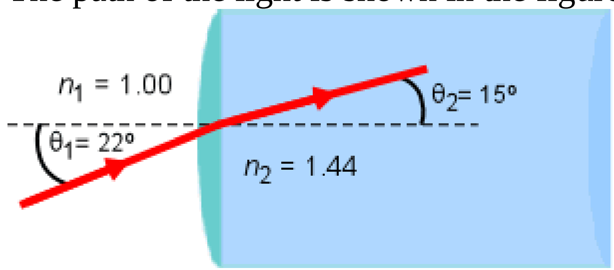
(b) We will identify air as medium 1 and the fiber as medium 2. Thus, $n_1 = 1.00$, $n_2 = 1.44$, and θ/font>$_1$ = 22°. Snell's Law then becomes

$$(1.00) \sin 22^{o} = 1.44 \sin \theta_2.$$

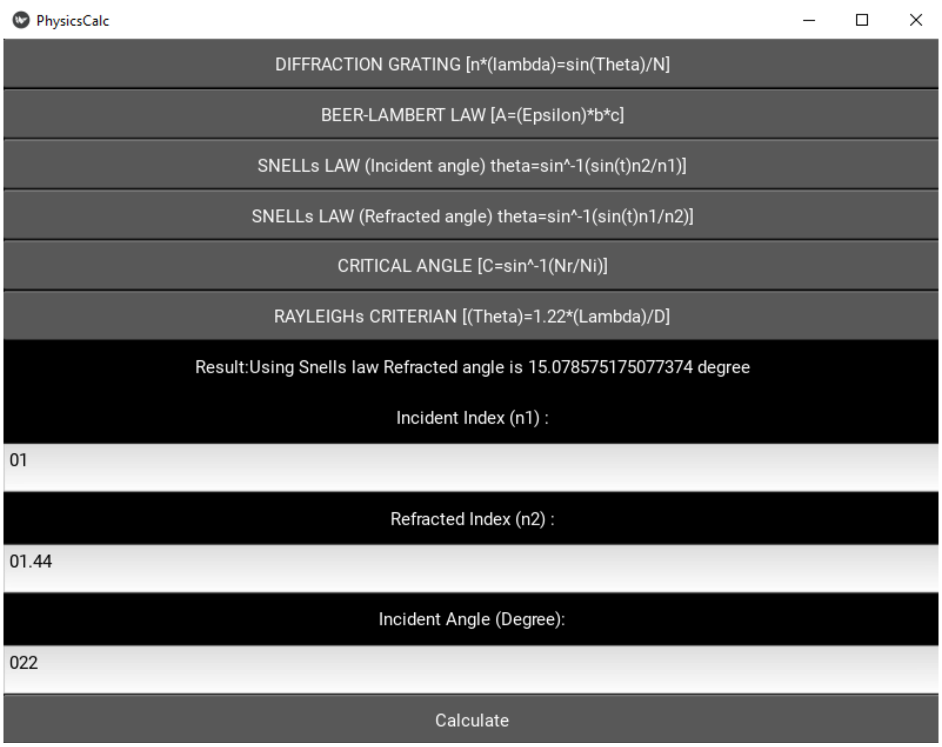
$$\sin \theta_2 = (1.00/1.44) \sin 22^{o} = 0.260$$

$$\theta_2 = \sin^{-1} (0.260) = 15^{o}.$$

(c) The path of the light is shown in the figure below.

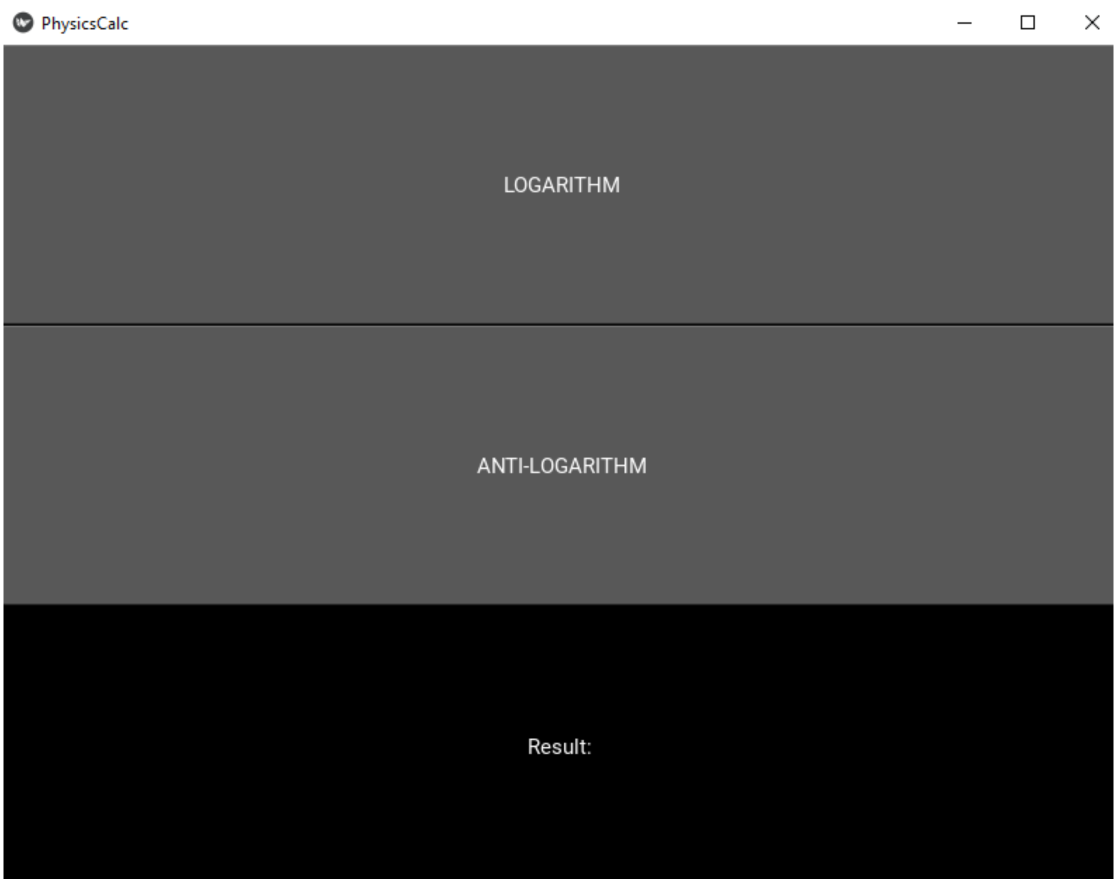


Calculations Result by application.



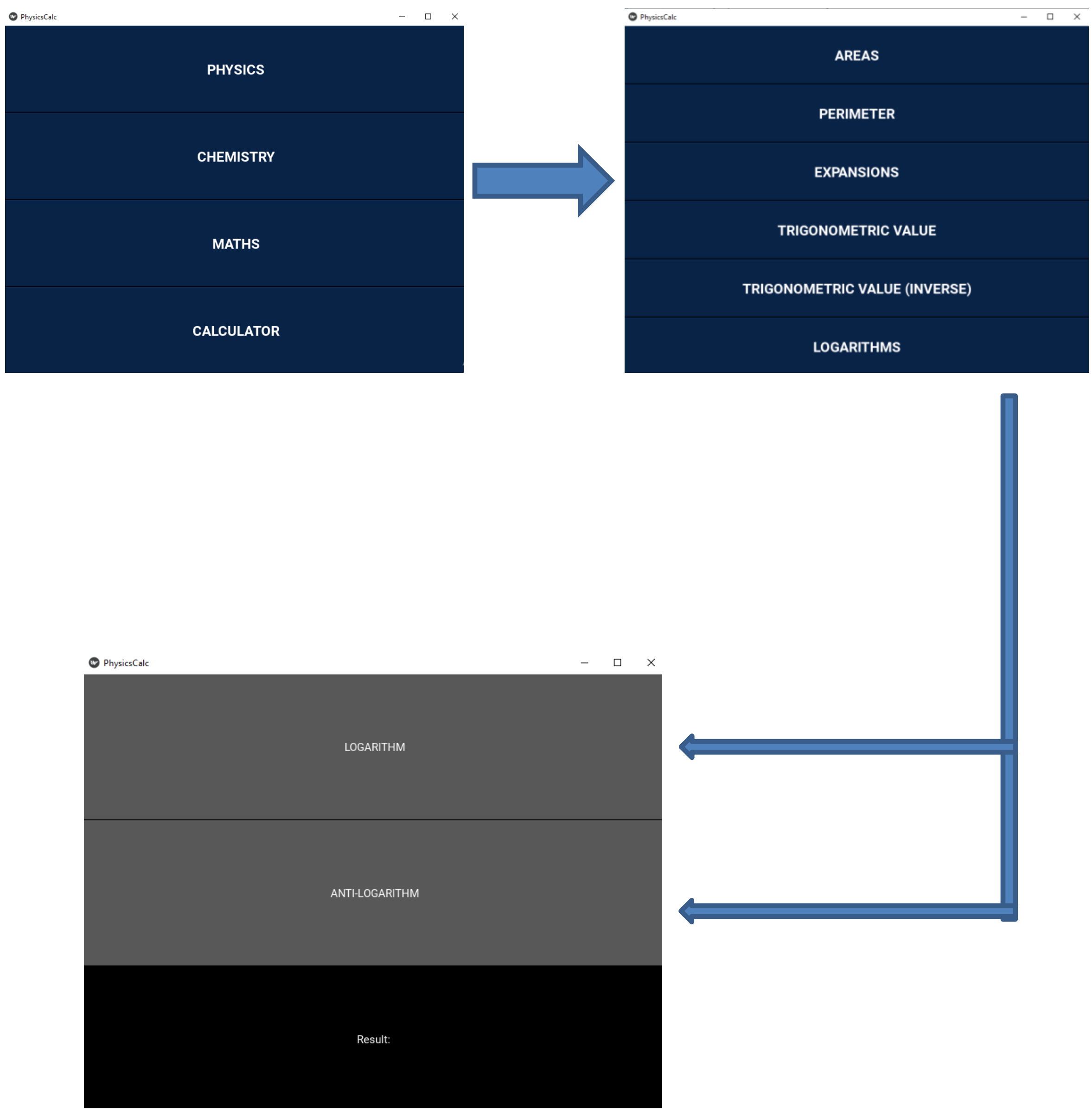
Result is: - 15.078575175077374 degree which is accurate and precise.

3. Logarithms & Anti-Logarithm



The application includes a dedicated page for solving logarithm and anti-logarithm problems with any desired base. This feature plays a significant role within the student community, as it is considered important and time-saving. Students appreciate the ability to quickly obtain accurate answers by utilizing this feature.

How to open Logarithm and Anti-logarithm page



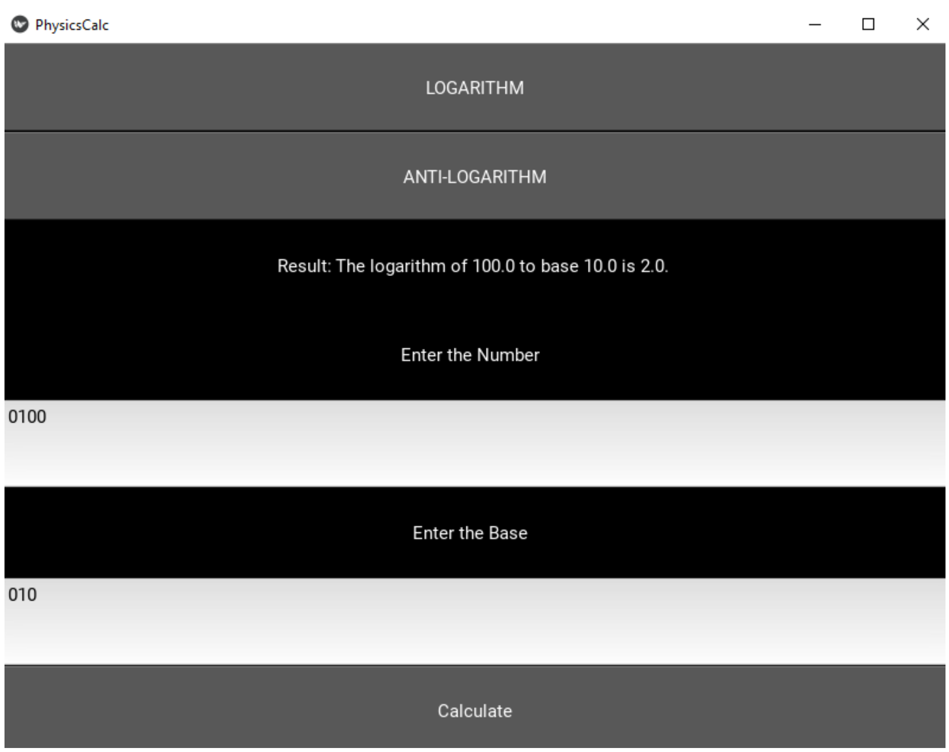
Example

Sample Problem 1:

Logarithm 100 to the base 10 is

Solution: 2.0

Calculations Result by application.

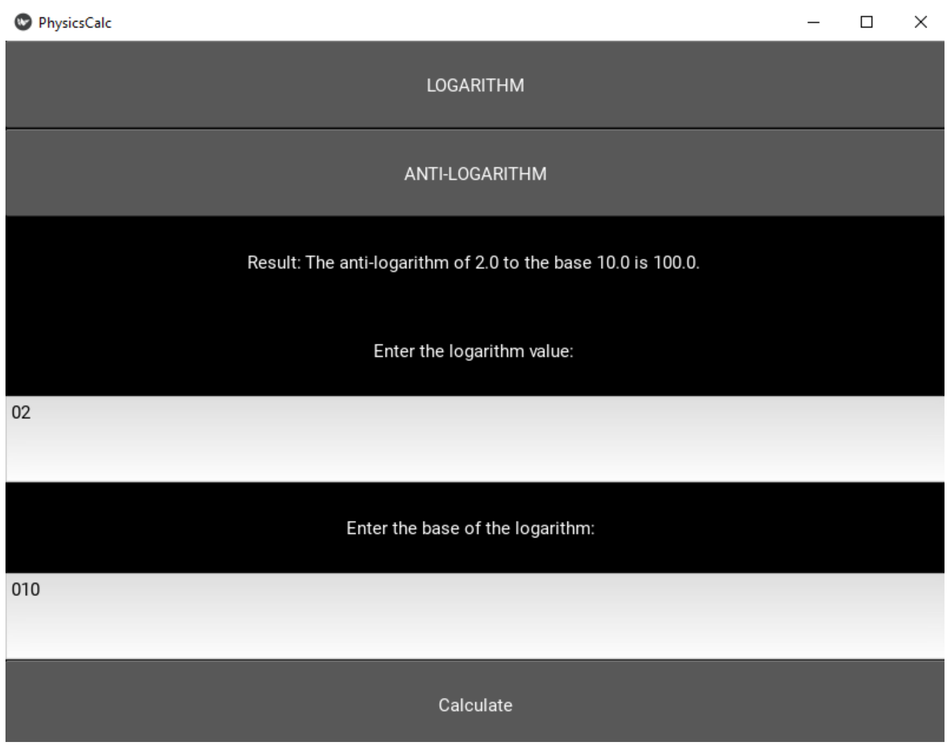


Sample Problem 2:

Anti-Logarithm 2 to the base 10 is

Solution: 100

Calculations Result by application.



CHAPATER 7:

FUTURE SCOPES

Here are some potential future scope ideas for our project:

1. Enhanced Functionality: We can expand the functionality of the scientific formula calculator by adding more subjects and formulas. Consider including additional scientific disciplines such as biology or astronomy, and include a broader range of formulas and calculations within each subject.
2. Customization Options: Allow users to customize the calculator interface by choosing different themes, colors, or layouts. This can enhance the user experience and make the application more visually appealing.
3. User Profiles and Data Storage: Implement user profiles to allow users to save their calculations, formulas, or favorite formulas for future reference. Additionally, you can provide the option to export or share calculations in various formats, such as PDF or Excel.
4. Offline Functionality: Enable the application to work offline by storing essential data and formulas locally on the user's device. This allows users to access and use the calculator even without an internet connection.
5. Unit Conversion: Incorporate a unit conversion feature that allows users to convert between different units of measurement. This can be particularly useful for scientific calculations that

involve different units, such as converting between metric and imperial units.

6. Interactive Tutorials: Provide interactive tutorials or step-by-step guides for complex calculations or concepts. This can help users understand and apply scientific formulas more effectively, especially for beginners or students.
7. Mobile App Development: Consider developing a mobile version of the application for iOS and Android devices. This allows users to access the calculator on their smartphones or tablets, making it more convenient and accessible.
8. Integration with External APIs or Databases: Connect the application to external APIs or databases to access real-time scientific data, such as chemical properties or astronomical data. This can provide users with up-to-date information and enhance the accuracy and relevance of calculations.
9. Collaboration and Sharing Features: Implement features that allow users to collaborate and share calculations or formulas with others. This can include features like real-time collaboration, sharing via email or social media, or creating public or private formula repositories.
10. Cross-Platform Compatibility: Ensure that the application is compatible with different operating systems and devices, such as Windows, macOS, Linux, and various web browsers. This expands the reach of your application and makes it accessible to a wider audience.

CHAPATER 8:

REFERANCES

1. Kivy Official website https://kivy.org/

2. Developing Apps for Android and Other Platforms with Kivy and Python by Andreas Schreiber

3. Kivy Documentation Release2.2.0rc1,https://buildmedia.readthedocs.org/media/pdf/kivy/latest/kivy.pdf

4. Creating Appliction in Kivy by Jesse lin, https://www.academia.edu/32422407/Creating_Apps_in_Kivy

5. Google.com