SASTA OLIPHANT SCIENCE AWARDS

Prize Winner

Programming, Apps & Robotics

Year 7-8

Jackson Burford

Aberfoyle Park High School

ROWE SCIENTIFIC PTY. LTD. www.rowe.com.au
Australian Government Department of Defence
Government of South Australia Department for Education
SASTA South Australian Science Teachers Association

Energy Conversion in Radioisotope Thermoelectric Generators

28/06/2024

By Jackson Burford

Introduction

RTG Stands for Radioisotope Thermoelectric Generator. An RTG is an invention for space travel to generate electricity in space for a long period of time, this can be from a few decades to potentially hundreds of years.

This is very important for deep space missions, as the RTG will provide reliable power for the instruments and the spacecraft.

This 2024 Oliphant Science Awards project is an RTG Mission Planner App. The user of the Mission Planner will give certain inputs to the App, (height and radius of the radioisotope cylinder used in the RTG, the isotope type, thermoelectric material and mission duration) and the app will use these inputs to calculate the power generated by the RTG over time.

Project Inspiration

This project is inspired by Sir Marcus Oliphant and his work with radioisotopes. In the 1930s, at Cavendish Laboratory, Sir Marcus Oliphant collaborated with Ernest Rutheford to bombard deuterium with deuterons which led to the discovery of two new isotopes, tritium and helium-3. Sir Marcus Oliphant also created new techniques for separating isotopes using electromagnetic fields. This was later used in WWII in the Manhattan Project. Sir Marcus Oliphant's work with hydrogen isotopes showed us the first experimental evidence of nuclear fusion. Sir Marcus Oliphant's contributions to isotope research were significant. He discovered new isotopes and developed methods for separating them.

About the Project

In this application you can choose out of two different isotopes, Plutonium-238 or Strontium-90.

Plutonium-238 (Pu-238) is used for space exploration, it is invented as fuel for an RTG, a radioisotope thermoelectric generator. Pu-238 is used as fuel because of its long half-life and its ability to generate heat.

Strontium-90 (Sr-90) is also a radioactive isotope created by nuclear fission; it can last about 28.8 years for half of its decay. Sr-90 generates heat as it decays, this makes it useful for space exploration.

An RTG uses the Seebeck effect to generate electricity, this is when you combine to conductors of electricity and applied heat to one end and exposing the other end to cold, an electrical voltage will be created across the materials, this is because electrons in the materials flow from the hot side to the cold side. These are referred to as a thermocouple and an RTG would use hundreds of thermocouples to generate electricity, and the isotopes decay heat which turns into the heat for the thermocouple.

Project Structure

- The project is organised into several Python files, each serving a specific purpose:
  - `main.py`: The entry point of the application, which initialises the GUI and starts the application.
  - `gui.py`: Manages the graphical user interface, including input fields, buttons, and tabs for different geometric configurations.
  - `calculations.py`: Contains the logic for calculating power output based on user inputs.
  - `plotting.py`: Handles the creation of interactive graphs using Plotly to visualise power output data.
  - `data.py`: Stores constants and data such as thermoelectric materials and isotope properties.
  - `utils.py`: Includes helper functions for input validation and exporting results.

How Application Works

- The user inputs various parameters into the application, such as isotope type, thermoelectric material, time elapsed, and load resistance.
- The application uses these inputs to calculate the power output of the RTG for a cylinder designed by the user.
- The calculations are based on the properties of the selected isotope and material, as well as the thermal and electrical equations governing RTG performance.
- The results are displayed on the GUI, and users can visualise the power output over time using an interactive graph.

What It Does

Once the user gives the required inputs, the application will calculate not only the power output, but the change in power output over time. The required inputs are the isotopes, thermoelectric materials, isotope cylinder radius, isotope cylinder height and mission duration.

Once given the inputs, you will also have access to a graph tab where you can see the power output of the RTG at the start of the mission and can see how the power output slowly goes down during the course of the mission.

The user will give inputs and then change the inputs to optimise the efficiency of the RTG.

Benefits

This app has many benefits, if advanced, could even be used in organisations like NASA and SpaceX, to help plan for their RTG Missions, or even without the advancements, could still be used as a practical tool for students and researchers to simulate and study RTG performance. This can also inspire students to get engaged in Space Exploration or Physics (Isotope studies).

It also enhances an understanding of RTG technology and the role of radioisotopes in space missions.

In the future, from near future to generations from now, deep space travel will become more and more common, and RTGs will be a very important part for generating electricity, as solar panels may not be as efficient when further from the sun.

Project History

The application started off with a simpler version that would only calculate the heat transfer rate (NO GUI OR GRAPH), however the difference with this version is that the user could choose whether to choose a cube, sphere, pyramid or cylinder, however I found out later that a cylinder is the shape that is most commonly used for an RTG.

Once the program was working it would display the heat transfer rate. At first it would display inaccurate outputs (One example in the screenshots). I went through the process of turning it into a GUI, the first version of the GUI was still very basic. It had a box at the top which would display the power output and under that were some of the required user inputs, height and radius. I did also have a few extra inputs such as DeltaT (Difference between hot side and cold side) and the Length of the wall of the Cylinder (this was assuming it was hollow, I removed this later because they are not supposed to be hollow). I still had the different shapes option for the first few versions of the GUI.

I upgraded the app so that it also displays power output power output (which was the original intention of the app as it was much more important than the heat transfer rate). To do this, I had to add extra inputs such as load resistance, isotope material and thermoelectric material.

It ended up being too hard to run in one large file of code so I used ChatGPT to help me separate the code into smaller files. (ChatGPT was used for cleaning up code, and helping troubleshoot final features). Once the code was cleaned, I also added a graph and a mission duration input so that the user could see the difference in power output over time.

The Future

For this app to be at a level where it could be useful in real space missions, there would have to be many changes, some possible things could potentially be to add more isotopes and more thermoelectric materials to the database.

It would also be important to enhance the user interface with more advanced features such as 3D visualisation of the RTGs and their components. Expanding the application to simulate other types of power systems used in space exploration.

A way to increase the simulation accuracy would be to integrate real-time data from space missions and collaborate with educational institutions to make the application a standard tool for science and engineering.

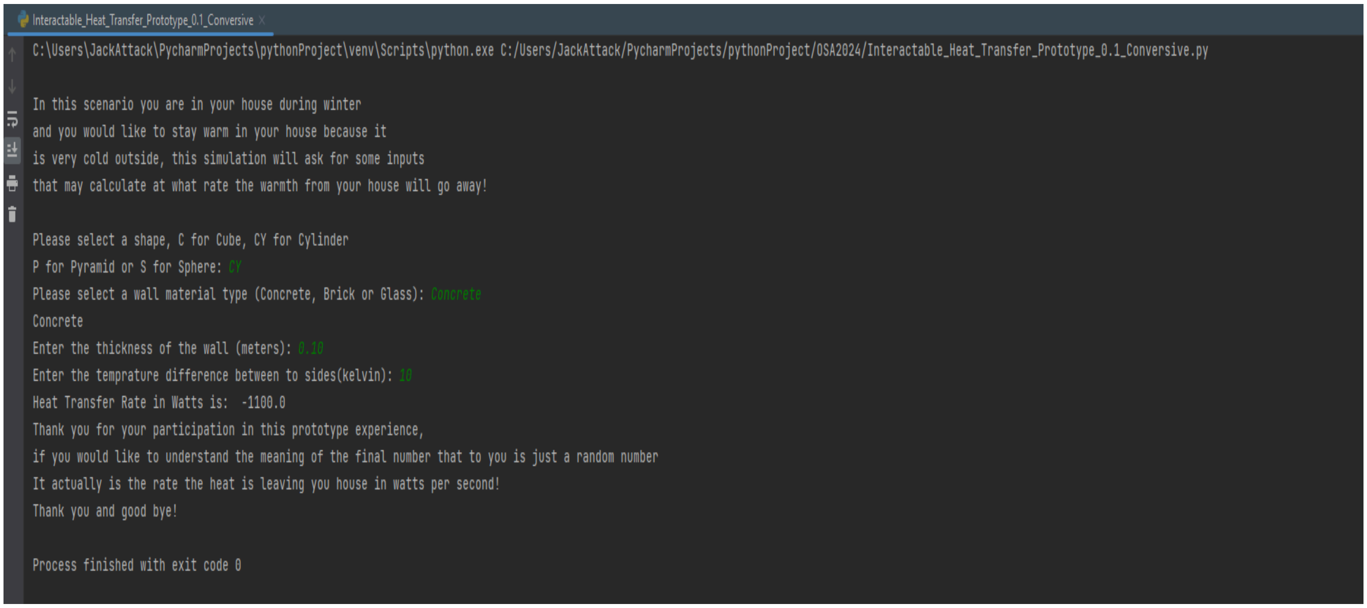
Conclusion

This application is combining software development with scientific research to create valuable educational tools. An RTG Mission planner is just one potential example of this, but this could span for many different things, and I see that they will only be more and more common in the future.

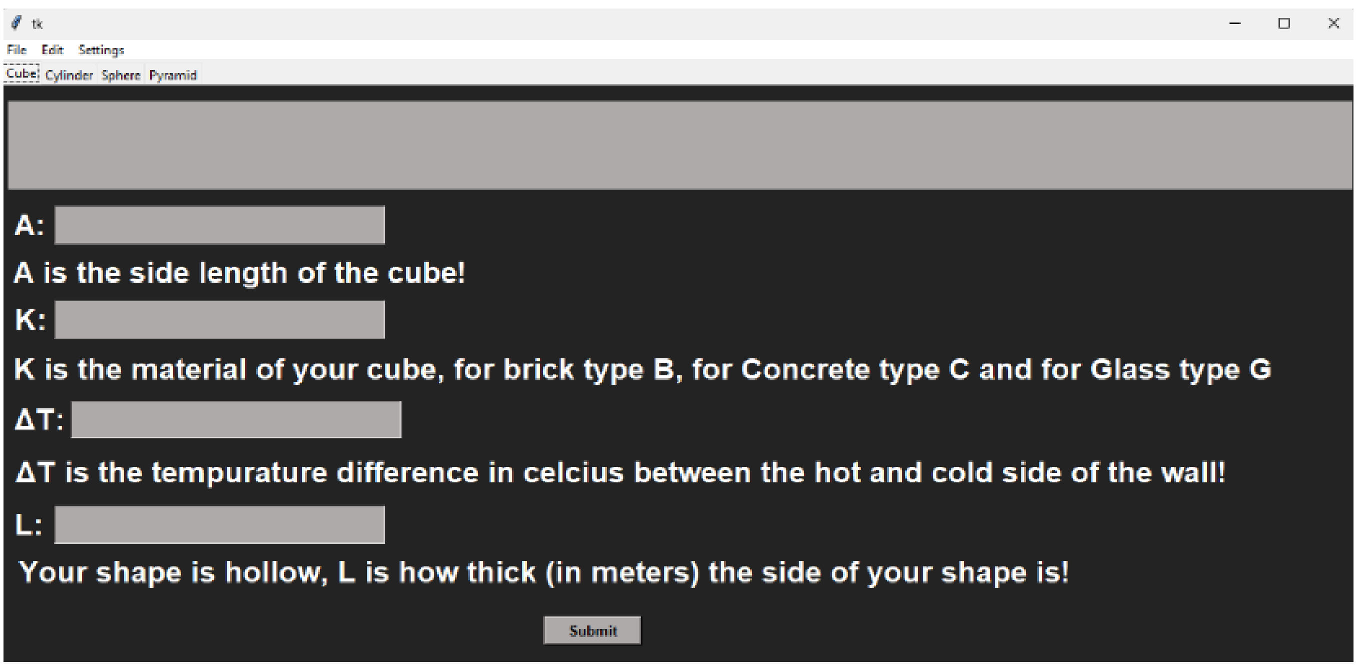
By simulation RTG performance, the user can learn about the crucial role of radioisotopes in space exploration. With Continued development, this application could become an app that is used for students, researchers and space mission planners.

This project is in honour of Marcus Oliphant and hopes to try and continue his legacy to inspire future generations to continue the work that he started.

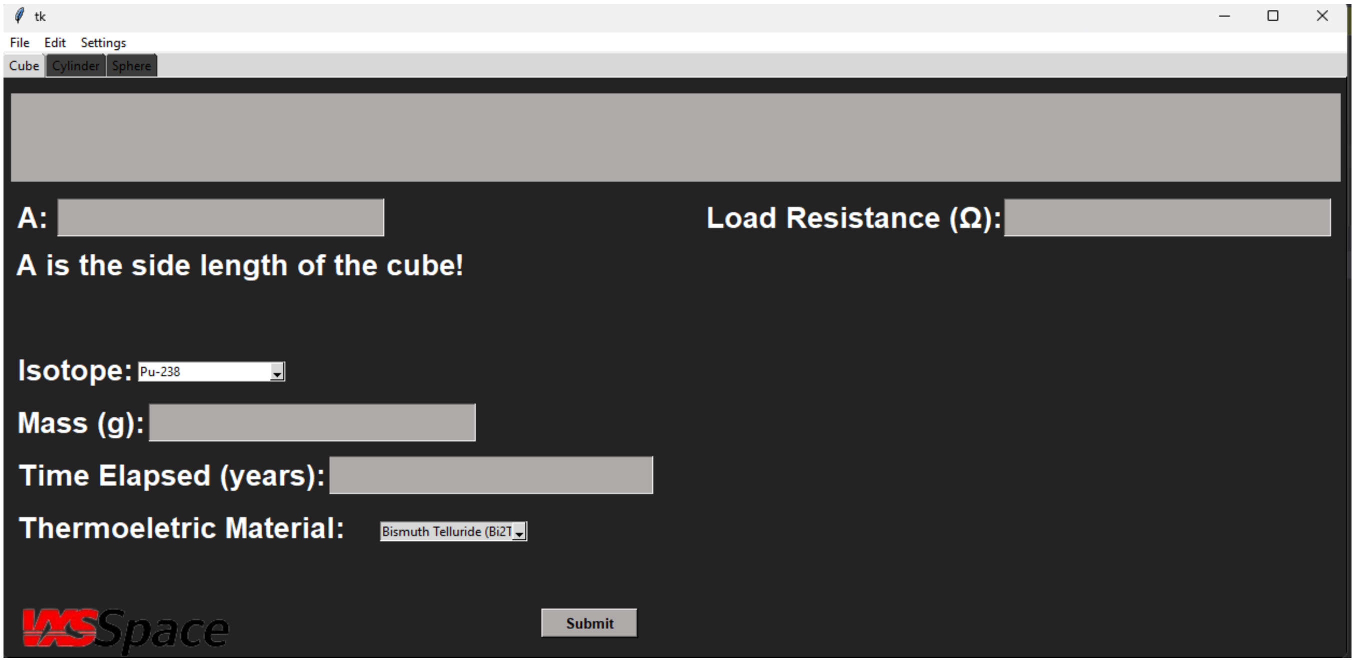
Screenshots



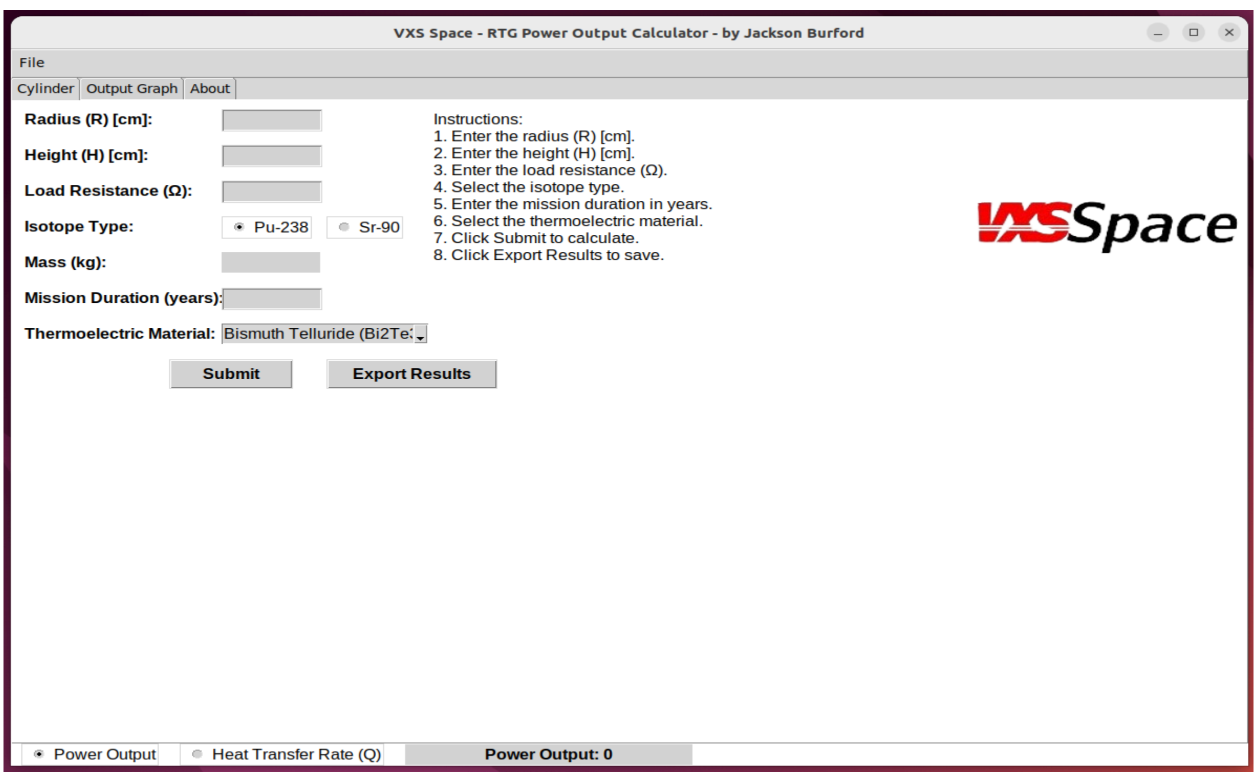
This is the very first version of my app, it was still missing necessary inputs and was only in the terminal. I was still using the concret, brick and grass material options and temperature difference in Kelvin.



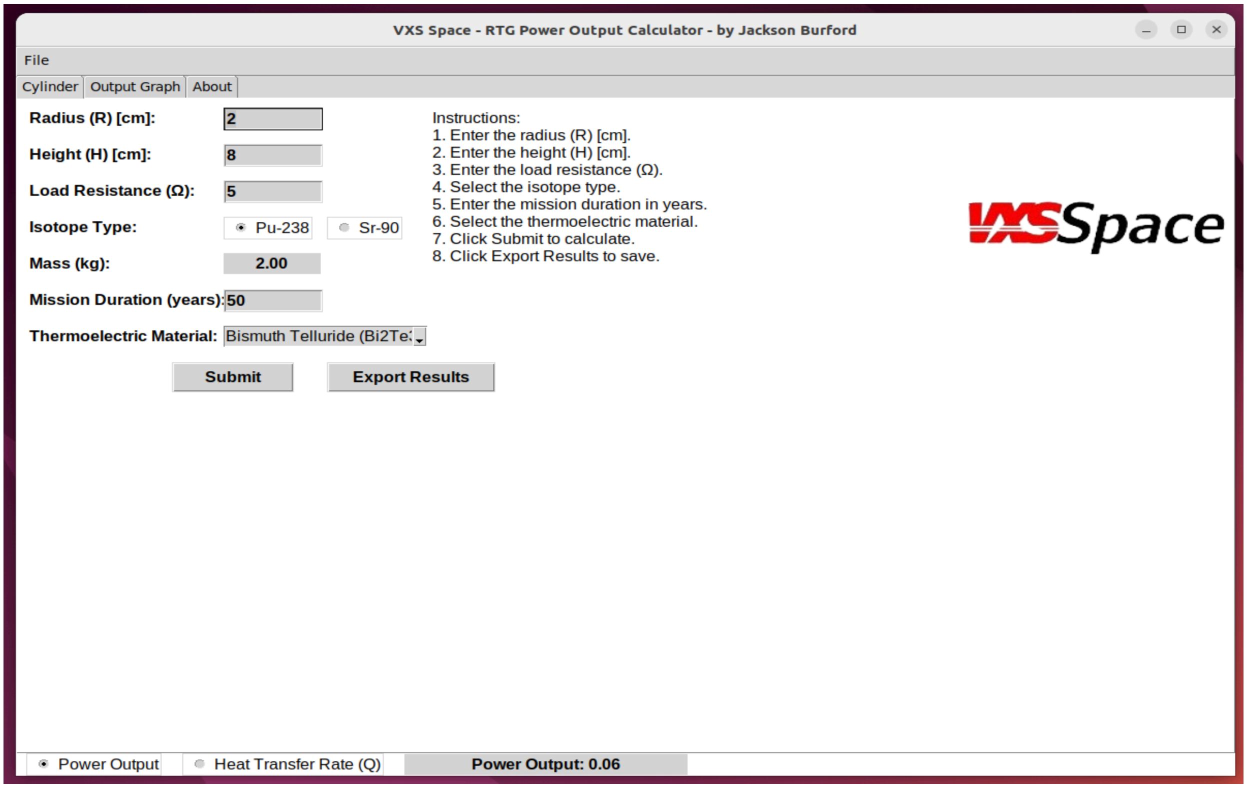
This is the first version of the GUI, works the same as the previous version except the temperature difference is now in celsius. The output would not work in this version.



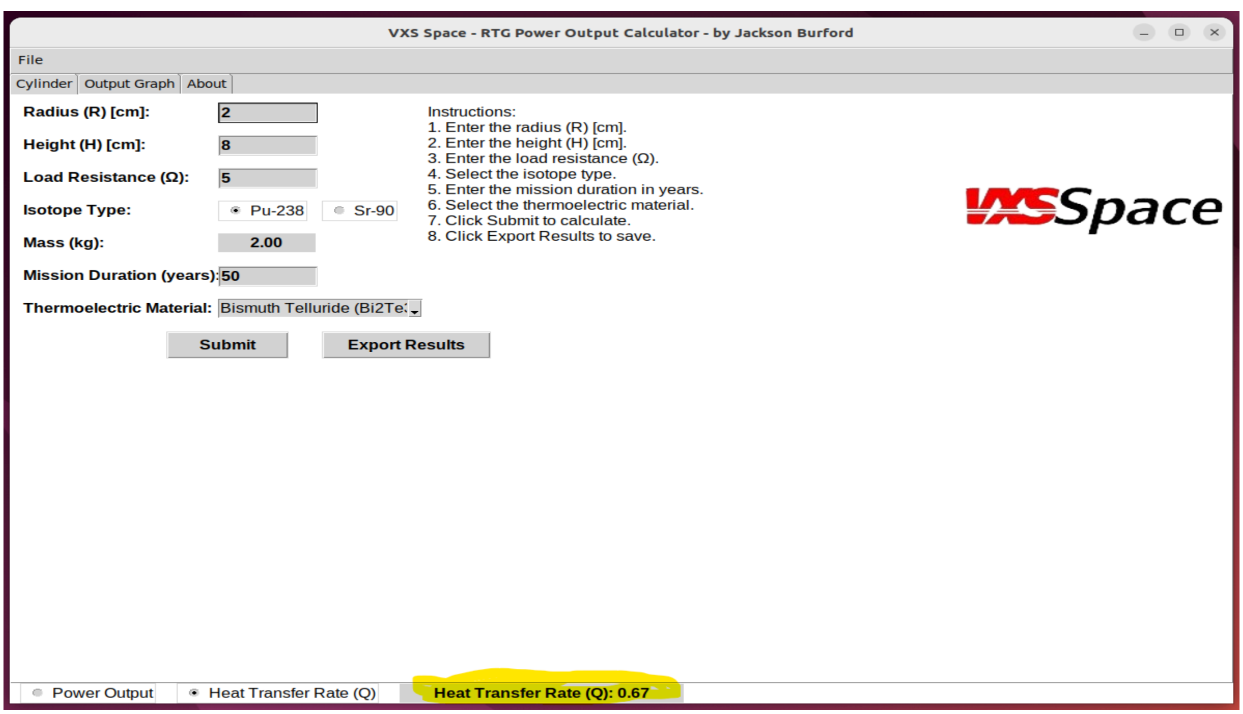
This is a later version of the GUI, whilst the design stayed similar, there were many differences. First of all, the pyramid tab was removed and the VXS Space Logo was added. The main part of the update was the new inputs which supported a more accurate output and allowed potential for power output calculation.



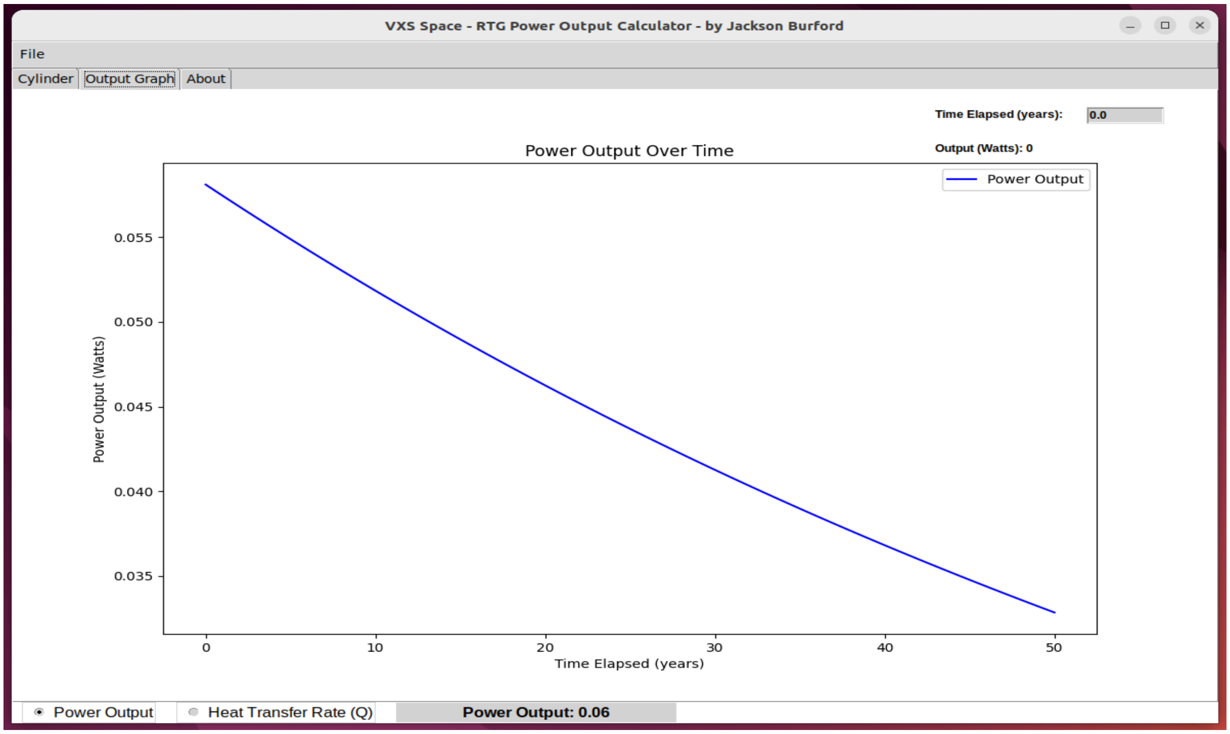
This is the newest design of the app, the cube and sphere tab was removed, it is now a light background instead of dark, and Mission duration has been added as an input, radius and height are now in cm, isotope type are now radio buttons, 2 new tabs, no edit or settings feature at the top, instructions and new output display. The main feature though is that it now shows power output.



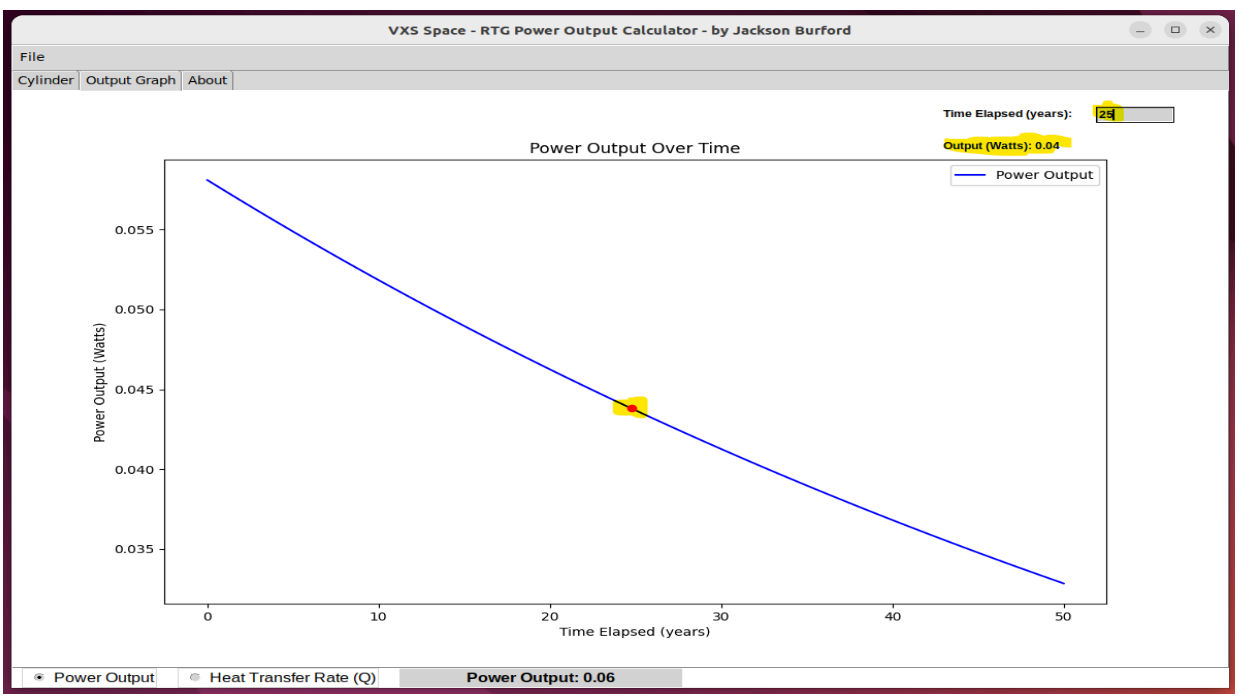
This is what the application looks like filled in (The Mass input has been removed since the screenshots have been taken because the Mass is calculated by Radius, Height and material.) The power output can be viewed in the grey box at the bottom.



This is what it looks like with the output option set on Heat Transfer Rate.



This is the output graph tab that shows the power output over time. You can see that the Y-Axis is the Power Output in watts while the X-Axis is the Time-Elapsed in years.



Here is the graph with the time elapsed option, you input how far you are in the mission and it shows the power output at that time and where on the graph you are.

Bibliography

https://www.youtube.com/watch?v=I-Puj0uyCAg

https://www.youtube.com/watch?v=wkVwWtRUqq4

https://science.nasa.gov/mission/cassini/radioisotope-thermoelectric-generator/

https://en.wikipedia.org/wiki/Radioisotope_thermoelectric_generator

https://science.nasa.gov/planetary-science/programs/radioisotope-power-systems/about-plutonium-238/

https://en.wikipedia.org/wiki/Plutonium-238

https://world-nuclear.org/information-library/nuclear-fuel-cycle/fuel-recycling/plutonium#:~:text=The%20decay%20heat%20of%20Pu,satellites%2C%20navigation%20beacons%2C%20etc.

https://www.ncbi.nlm.nih.gov/books/NBK599413/table/ch1.tab1/#:~:text=The%20half%2Dlife%20is%20the,plutonium%2D238%20is%2087.7%20years.

https://en.wikipedia.org/wiki/Strontium-90#:~:text=Uses-,Radioisotope%20thermoelectric%20generators%20(RTGs),than%20the%20alternative%20238Pu.

https://dhss.delaware.gov/dhss/dph/files/strontiumfaq.pdf

https://www.sciencedirect.com/science/article/abs/pii/S0378775322003172#:~:text=Bismuth%20telluride%2Dbased%20compounds%20are,efficiency%20(3%E2%80%936%25).

https://pubs.rsc.org/en/content/articlelanding/2022/ee/d1ee03883d#:~:text=Fabricated%20single%20and%20segmented%20thermoelectric,%25%20and%2012.2%25%2C%20respectively.

https://www.sciencedirect.com/science/article/abs/pii/S2542529324001421

https://www.sciencedirect.com/science/article/pii/S2214785320344217

NOTE: USED PYTHON TUTORIALS ON UDEMY FOR LEARNING PYTHON

APPENDIX

main.py:

```
from tkinter import Tk
from gui import create_gui

def main():
    window = Tk()
    window.title("VXS Space - RTG Power Output Calculator - by Jackson
Burford")
    window.geometry("1200x800")

    create_gui(window)

    window.mainloop()

if __name__ == "__main__":
    main()
```

plotting.py:

```
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

def embed_plot_to_tk(parent, width=5, height=4, dpi=100):
    fig = plt.Figure(figsize=(width, height), dpi=dpi)
    ax = fig.add_subplot(111)
    canvas = FigureCanvasTkAgg(fig, master=parent)
    canvas.draw()
    canvas.get_tk_widget().pack(side="top", fill="both", expand=True)
    return ax

def update_plot(ax, time_elapsed_series, power_output_series,
highlight_index=None):
    ax.clear()
    ax.plot(time_elapsed_series, power_output_series, label='Power Output',
color='blue')
```

```
    if highlight_index is not None:
        ax.plot(time_elapsed_series[highlight_index],
power_output_series[highlight_index], 'ro')  # Highlight the selected point
    ax.set_xlabel("Time Elapsed (years)")
    ax.set_ylabel("Power Output (Watts)")
    ax.set_title("Power Output Over Time")
    ax.legend()
    ax.figure.canvas.draw()
```

data.py:

```
isotope_data = {
    "Pu-238": {"half_life": 87.7, "thermal_power_per_gram": 0.57},
    "Sr-90": {"half_life": 29.0, "thermal_power_per_gram": 0.95},
}

thermoelectric_data = {
    "Bismuth Telluride (Bi2Te3)": {"efficiency": 0.03},
    "Lead Telluride (PbTe)": {"efficiency": 0.093},
    "Germanium Telluride (GeTe)": {"efficiency": 0.10},
    "Skutterudite Alloys": {"efficiency": 0.12},
    "Other": {},
}

isotopes = ["Pu-238", "Sr-90"]
```

calculations.py:

```
import numpy as np
from data import isotope_data, thermoelectric_data

def submitk(size1, size2, load_resistance, isotope_type, mass,
thermoelectric_material, time_elapsed):
    # Calculate the volume for a cylinder
    volume = np.pi * (size1 / 100) ** 2 * (size2 / 100)  # Convert cm to m

    # Get isotope data
    isotope = isotope_data[isotope_type]
```

```
    half_life = isotope["half_life"]
    thermal_power_per_gram = isotope["thermal_power_per_gram"]

    # Calculate remaining mass after time elapsed
    remaining_mass = mass * (0.5 ** (time_elapsed / half_life))

    # Calculate thermal power
    thermal_power = remaining_mass * thermal_power_per_gram

    # Get thermoelectric material efficiency
    efficiency = thermoelectric_data.get(thermoelectric_material,
{}).get("efficiency", None)
    if efficiency is None:
        raise ValueError(f"Efficiency for thermoelectric material
'{thermoelectric_material}' is not defined.")

    # Calculate power output
    power_output = thermal_power * efficiency

    # Calculate heat transfer rate (Q)
    heat_transfer_rate = thermal_power * (1 - efficiency)

    # Generate time series for the graph
    time_elapsed_series = np.linspace(0, time_elapsed, 100)
    power_output_series = power_output * np.exp(-time_elapsed_series /
half_life)

    return time_elapsed_series, power_output_series, power_output,
heat_transfer_rate
```

gui.py:

```
from tkinter import Frame, Label, Entry, Button, Menu, StringVar, ttk, LEFT,
Radiobutton, DoubleVar, messagebox, TclError, PhotoImage
from calculations import submitk
from data import isotopes, thermoelectric_data
from plotting import embed_plot_to_tk, update_plot
from about import create_about_tab
import math
```

```
# Define the densities of the isotopes in g/cm³
isotope_densities = {
    "Pu-238": 19.86,   # g/cm³
    "Sr-90": 2.8,      # g/cm³ (approximate value, actual value may vary)
}

def create_gui(window):
    global tabCylinder, tabGraph, notebook, output_var, output_label, entryr,
entryh, entry_resistance2, isotope_var2, entry_time2, thermoelectric_var2,
power_output, heat_transfer_rate, time_elapsed_series, power_output_series,
dynamic_time_var
    notebook = ttk.Notebook(window)
    s = ttk.Style()
    s.theme_use('default')
    s.configure('TNotebook.Tab', background="#d3d3d3")
    s.map("TNotebook", background=[("selected", "#d3d3d3")])

    tabCylinder = Frame(notebook, background="#ffffff")
    tabGraph = Frame(notebook, background="#ffffff")
    tabAbout = Frame(notebook, background="#ffffff")

    notebook.add(tabCylinder, text="Cylinder")
    notebook.add(tabGraph, text="Output Graph")
    notebook.add(tabAbout, text="About")
    notebook.pack(expand=True, fill="both")

    menubar = Menu(window)
    window.config(menu=menubar)

    fileMenu = Menu(menubar, tearoff=0, font=("Arial", 10))
    menubar.add_cascade(label="File", menu=fileMenu)
    fileMenu.add_separator()
    fileMenu.add_command(label="Exit", command=window.quit)

    # Output selection and label at the top
    output_frame = Frame(window, bg='#ffffff')
    output_frame.pack(side="top", fill="x")
```

```
    output_var = StringVar(window)
    output_var.set("Power Output")
    power_output_radio = Radiobutton(output_frame, text="Power Output",
variable=output_var, value="Power Output", font=("Arial", 12), bg='#ffffff',
fg='black', selectcolor='#d3d3d3', command=update_output)
    power_output_radio.pack(side="left", padx=10)
    heat_transfer_radio = Radiobutton(output_frame, text="Heat Transfer Rate
(Q)", variable=output_var, value="Heat Transfer Rate (Q)", font=("Arial",
12), bg='#ffffff', fg='black', selectcolor='#d3d3d3', command=update_output)
    heat_transfer_radio.pack(side="left", padx=10)

    output_label = Label(output_frame, text="Power Output: 0", font=("Arial",
12, 'bold'), width=30, height=1, background="#d3d3d3")
    output_label.pack(side="left", padx=10)

    create_cylinder_tab(tabCylinder)
    create_graph_tab(tabGraph)
    create_about_tab(tabAbout)

def create_cylinder_tab(tab):
    global entryr, entryh, entry_time2, entry_mass2, entry_resistance2,
isotope_var2, thermoelectric_var2

    # Input Fields
    rlabel = Label(tab, text="Radius (R) [cm]:", font=('Arial', 12, 'bold'),
fg='black', bg='#ffffff')
    rlabel.place(x=10, y=10)
    entryr = Entry(tab, font=("Arial", 12, 'bold'), fg="black", bg="#d3d3d3",
width=10)
    entryr.place(x=200, y=10)

    helabel = Label(tab, text="Height (H) [cm]:", font=('Arial', 12, 'bold'),
fg='black', bg='#ffffff')
    helabel.place(x=10, y=50)
    entryh = Entry(tab, font=("Arial", 12, 'bold'), fg="black", bg="#d3d3d3",
width=10)
    entryh.place(x=200, y=50)
```

```
    load_resistance_label2 = Label(tab, text="Load Resistance (Ω):",
font=('Arial', 12, 'bold'), fg='black', bg='#ffffff')
    load_resistance_label2.place(x=10, y=90)
    entry_resistance2 = Entry(tab, font=("Arial", 12, 'bold'), fg="black",
bg="#d3d3d3", width=10)
    entry_resistance2.place(x=200, y=90)

    isotope_label2 = Label(tab, text="Isotope Type:", font=('Arial', 12,
'bold'), fg='black', bg='#ffffff')
    isotope_label2.place(x=10, y=130)
    isotope_var2 = StringVar(tab)
    isotope_var2.set(isotopes[0])
    isotope_radio3 = Radiobutton(tab, text="Pu-238", variable=isotope_var2,
value="Pu-238", font=("Arial", 12), bg='#ffffff', fg='black',
selectcolor='#d3d3d3')
    isotope_radio3.place(x=200, y=130)
    isotope_radio4 = Radiobutton(tab, text="Sr-90", variable=isotope_var2,
value="Sr-90", font=("Arial", 12), bg='#ffffff', fg='black',
selectcolor='#d3d3d3')
    isotope_radio4.place(x=300, y=130)

    mass_label2 = Label(tab, text="Mass (kg):", font=('Arial', 12, 'bold'),
fg='black', bg='#ffffff')
    mass_label2.place(x=10, y=170)
    entry_mass2 = Label(tab, font=("Arial", 12, 'bold'), fg="black",
bg="#d3d3d3", width=10)
    entry_mass2.place(x=200, y=170)

    time_label2 = Label(tab, text="Mission Duration (years):", font=('Arial',
12, 'bold'), fg='black', bg='#ffffff')
    time_label2.place(x=10, y=210)
    entry_time2 = Entry(tab, font=("Arial", 12, 'bold'), fg="black",
bg="#d3d3d3", width=10)
    entry_time2.place(x=200, y=210)

    thermoelectric_label2 = Label(tab, text="Thermoelectric Material:",
font=('Arial', 12, 'bold'), fg='black', bg='#ffffff')
    thermoelectric_label2.place(x=10, y=250)
    thermoelectric_var2 = StringVar(tab)
```

```
    thermoelectric_dropdown2 = ttk.Combobox(tab,
textvariable=thermoelectric_var2, values=list(thermoelectric_data.keys()),
state="readonly", font=("Arial", 12))
    thermoelectric_dropdown2.place(x=200, y=250)
    thermoelectric_dropdown2.current(0)

    # Submit Button
    submit_button2 = Button(tab, text="Submit", font=("Arial", 12, 'bold'),
command=submit_and_plot_cylinder, height=1, width=10, bg="#d3d3d3",
fg="black")
    submit_button2.place(x=150, y=290)

    # Export Results Button
    export_button2 = Button(tab, text="Export Results", font=("Arial", 12,
'bold'), command=export_results, height=1, width=15, bg="#d3d3d3",
fg="black")
    export_button2.place(x=300, y=290)

    # Instructions
    instructions2 = Label(tab, text="Instructions:\n1. Enter the radius (R)
[cm].\n2. Enter the height (H) [cm].\n3. Enter the load resistance (Ω).\n4.
Select the isotope type.\n5. Enter the mission duration in years.\n6. Select
the thermoelectric material.\n7. Click Submit to calculate.\n8. Click Export
Results to save.", font=('Arial', 12), fg='black', bg='#ffffff',
justify=LEFT)
    instructions2.place(x=400, y=10)

    # Add the logo
    logo_img = PhotoImage(file="logo.png")
    logo_label = Label(tab, image=logo_img, bg='#ffffff')
    logo_label.image = logo_img
    logo_label.place(relx=1.0, y=0, anchor="ne", x=-5)

def create_graph_tab(tab):
    global ax_graph, dynamic_time_var, dynamic_output_label
    ax_graph = embed_plot_to_tk(tab, width=5.5, height=4)

    dynamic_time_label = Label(tab, text="Time Elapsed (years):",
font=('Arial', 9, 'bold'), fg='black', bg='#ffffff')
```

```
    dynamic_time_label.place(x=900, y=20)
    dynamic_time_var = DoubleVar()
    dynamic_time_entry = Entry(tab, textvariable=dynamic_time_var,
font=("Arial", 9, 'bold'), fg="black", bg="#d3d3d3", width=10)
    dynamic_time_entry.place(x=1050, y=20)
    dynamic_time_var.trace_add('write', lambda *args: update_dynamic_point())

    dynamic_output_label = Label(tab, text="Output (Watts): 0",
font=('Arial', 9, 'bold'), fg='black', bg='#ffffff')
    dynamic_output_label.place(x=900, y=60)


def submit_and_plot_cylinder():
    global power_output, heat_transfer_rate, time_elapsed_series,
power_output_series
    if not validate_inputs_cylinder():
        return

    radius = float(entryr.get())
    height = float(entryh.get())
    load_resistance = float(entry_resistance2.get())
    isotope_type = isotope_var2.get()
    time_elapsed = float(entry_time2.get())
    thermoelectric_material = thermoelectric_var2.get()

    # Calculate the mass based on the radius, height, and selected isotope
    volume = math.pi * radius**2 * height  # cm³
    density = isotope_densities[isotope_type]  # g/cm³
    mass = volume * density  # g
    mass_kg = mass / 1000  # kg

    entry_mass2.config(text=f"{mass_kg:.2f}")  # Update the mass field

    try:
        time_elapsed_series, power_output_series, power_output,
heat_transfer_rate = submitk(radius, height, load_resistance, isotope_type,
mass_kg, thermoelectric_material, time_elapsed)
    except ValueError as e:
        messagebox.showerror("Error", str(e))
```

```
        return

    # Switch to the Output Graph tab
    notebook.select(tabGraph)

    # Display the selected output in the output field
    update_output()

    update_plot(ax_graph, time_elapsed_series, power_output_series)

def validate_inputs_cylinder():
    if not entryr.get() or not entryh.get() or not entry_resistance2.get() or
not entry_time2.get():
        messagebox.showwarning("Input Error", "Please fill in all required
fields.")
        return False
    return True

def update_output():
    selected_output = output_var.get()
    if selected_output == "Power Output":
        output_value = power_output
    else:
        output_value = heat_transfer_rate

    output_label.config(text=f"{selected_output}: {output_value:.2f}")

def update_output_graph():
    selected_output = output_var.get()
    update_plot(ax_graph, time_elapsed_series, power_output_series)

def update_dynamic_point():
    global time_elapsed_series, power_output_series, dynamic_output_label
    try:
        time_elapsed = dynamic_time_var.get()
    except TclError:
        dynamic_output_label.config(text="Output (Watts): N/A")
        return
```

```
    if 0 <= time_elapsed <= max(time_elapsed_series):
        index = int(time_elapsed / max(time_elapsed_series) *
(len(time_elapsed_series) - 1))
        dynamic_output = power_output_series[index]
        dynamic_output_label.config(text=f"Output (Watts):
{dynamic_output:.2f}")
        update_plot(ax_graph, time_elapsed_series, power_output_series,
highlight_index=index)
    else:
        dynamic_output_label.config(text="Output (Watts): N/A")

def export_results():
    from utils import get_simulation_results, export_to_csv
    data = get_simulation_results()  # This function should gather the
current results
    export_to_csv(data)
```

OLD ORIGINAL CODE:

```
from tkinter import *
from tkinter import ttk
from math import pi
from math import *
import math
import time

T_h = 600
T_c = -273

DeltaT = T_c - T_h

#(DeltaT)

thermoelectric_data = {
    "Bismuth Telluride (Bi2Te3)": {"efficiency": 0.08},  # Efficiency example
```

```
   "Lead Telluride (PbTe)": {"efficiency": 0.07},  # Efficiency example
   "Germanium Telluride (GeTe)": {"efficiency": 0.05},  # Example material
   "Skutterudite Alloys": {"efficiency": 0.12},  # Example material
   "Other": {},  # Placeholder for user-defined materials
}



isotope_data = {
 "Pu-238": {"decay_constant": 0.018, "energy_per_decay": 5.59e-12},

 "Sr-90": {"decay_constant": 0.0281, "energy_per_decay": 2.83e-11},
}



Pi = pi

program_start_time = time.time()

def submitk():
   global K
   global eta_th
   global alpha
   global DeltaT
   K = 400



   isotope_type = isotope_var.get()

   selected_material = thermoelectric_var.get()

   eta_th = float(thermoelectric_data[selected_material]["efficiency"])



   thermo = {
   "Bismuth Telluride (Bi2Te3)": {"efficiency": 0.08},  # Efficiency example
```

```
    "Lead Telluride (PbTe)": {"efficiency": 0.07},  # Efficiency example
    "Germanium Telluride (GeTe)": {"efficiency": 0.05},  # Example material
    "Skutterudite Alloys": {"efficiency": 0.12},  # Example material
    "Other": {},
    }

    #print(eta_th)

    alpha = thermoelectric_data[selected_material][
    "efficiency"] * 1e-3  # Convert efficiency to Seebeck coefficient (V/K)

    try:
        time_elapsed = float(entry_time.get())
        # Perform validation (e.g., check if time is positive)
        if time_elapsed <= 0:
            raise ValueError("Time elapsed must be a positive number")
    except ValueError as e:
        print("Error:", e)
        # You can display an error message on the GUI using a label or popup
window
        return

    # Get initial mass from user input
    try:
        mass = float(entry_mass.get())
        # Perform validation (e.g., check if mass is positive)
        if mass <= 0:
            raise ValueError("Mass must be a positive number")
    except ValueError as e:
        print("Error:", e)
        # You can display an error message on the GUI using a label or popup
window
        return

    try:
        load_resistance = float(entry_resistance.get())
        # Perform validation (e.g., check if resistance is positive)
        if load_resistance <= 0:
            raise ValueError("Load resistance must be a positive number")
```

```
    except ValueError as e:
        print("Error:", e)
        # You can display an error message on the GUI using a label or popup
window
        return

    eta_C = 1 - (T_c / (T_h + 273.15))

    effective_efficiency = eta_C * eta_th

    #test_rtg_calculator()

    # Get time elapsed from user input (replace with your implementation)  #
Assuming you have a way to get time from the user (e.g., Entry field,
Spinbox)

    # Access decay data based on isotope type
    decay_constant = isotope_data[isotope_type]["decay_constant"]
    energy_per_decay = isotope_data[isotope_type]["energy_per_decay"]

    # Calculate decay factor
    decay_factor = math.exp(-decay_constant * time_elapsed)

    # Calculate heat output directly
    heat_decay_watts = mass * decay_constant * energy_per_decay *
decay_factor  # Convert Joules per year to Watts

    ta_ht = 0.8
    heat_decay_rate = heat_decay_watts * ta_ht


    deltaT = 873

    a = int(entrya.get())

    A = 6*(a*a)

    L = a/2
```

```
    Q = -K * A * deltaT / L + heat_decay_watts

    #Q2 = -K * A * deltaT / L + heat_decay_watts

    #print(Q2)

    #print(heat_decay_watts)

#     print(Q)

    #ANSlabel.config(text=Q)


    heat_transfer_rate = Q

    #estimated_power = heat_transfer_rate * effective_efficiency

    delta_t = float(deltaT)

    # Calculate voltage using Seebeck coefficient (α) and ΔT
    voltage = eta_th * delta_t

    # Power output using Joule's Law (final value)
    power_output = voltage ** 2 / load_resistance

    #ANSlabel.config(power_output)
    print(power_output)

#def submitkb():
#     submitk(0)

#materials = {"Concrete": 1.1, "Brick": 1.0, "Glass": 0.8}

def submitcy():
    global Kcy
    Kcy = 400
    #user_input = entryk
    #K = user_input
```

```
    isotope_type = isotope_var.get()

    try:
        time_elapsed = float(entry_time2.get())
        # Perform validation (e.g., check if time is positive)
        if time_elapsed <= 0:
            raise ValueError("Time elapsed must be a positive number")
    except ValueError as e:
        print("Error:", e)
        # You can display an error message on the GUI using a label or popup
window
        return

    # Get initial mass from user input
    try:
        mass = float(entry_mass2.get())
        # Perform validation (e.g., check if mass is positive)
        if mass <= 0:
            raise ValueError("Mass must be a positive number")
    except ValueError as e:
        print("Error:", e)
        # You can display an error message on the GUI using a label or popup
window
        return

    # Get time elapsed from user input (replace with your implementation)  #
Assuming you have a way to get time from the user (e.g., Entry field,
Spinbox)

    # Access decay data based on isotope type
    decay_constant = isotope_data[isotope_type]["decay_constant"]
    energy_per_decay = isotope_data[isotope_type]["energy_per_decay"]

    # Calculate decay factor
    decay_factor = math.exp(-decay_constant * time_elapsed)

    # Calculate heat output directly
```

```
    heat_decay_watts = mass * decay_constant * energy_per_decay *
decay_factor

    #deltaT = int(entryt.get())

    deltaTcy = 873

    radius = int(entryr.get())
    h = int(entryh.get())

    sacy = (2*pi*(radius*radius)) + (2*pi*radius*h)

    Lcy = radius/2

    Qcy = -Kcy * sacy * deltaTcy / Lcy + heat_decay_watts

    print(Qcy)

    ANSlabelcy.config(text=Qcy)

def submits():
    global Ks
    Ks = 400
    deltaTs = 873

    isotope_type = isotope_var.get()

    try:
        time_elapsed = float(entry_time3.get())
        # Perform validation (e.g., check if time is positive)
        if time_elapsed <= 0:
            raise ValueError("Time elapsed must be a positive number")
    except ValueError as e:
        print("Error:", e)
        # You can display an error message on the GUI using a label or popup
window
        return

    # Get initial mass from user input
```

```
    try:
        mass = float(entry_mass3.get())
        # Perform validation (e.g., check if mass is positive)
        if mass <= 0:
            raise ValueError("Mass must be a positive number")
    except ValueError as e:
        print("Error:", e)
        # You can display an error message on the GUI using a label or popup
window
        return

    # Get time elapsed from user input (replace with your implementation)  #
Assuming you have a way to get time from the user (e.g., Entry field,
Spinbox)

    # Access decay data based on isotope type
    decay_constant = isotope_data[isotope_type]["decay_constant"]
    energy_per_decay = isotope_data[isotope_type]["energy_per_decay"]

    # Calculate decay factor
    decay_factor = math.exp(-decay_constant * time_elapsed)

    # Calculate heat output directly
    heat_decay_watts = mass * decay_constant * energy_per_decay *
decay_factor

    r = int(entryra.get())

    sra = 4*pi*(r*r)

    Lra = r/2

    Qra = -Ks * sra * deltaTs / Lra + heat_decay_watts

    print(Qra)

    ANSlabels.config(text=Qra)

window = Tk()
```

```
window.geometry("1250x575")
window.config(background="#404040")

notebook = ttk.Notebook(window)


s = ttk.Style()
s.theme_use('default')
s.configure('TNotebook.Tab', background="#404040")
s.map("TNotebook", background= [("selected", "#404040")])










tabCube = Frame(notebook, background="#262626")
tabCylinder = Frame(notebook,background="#262626")
tabSphere = Frame(notebook, background="#262626")


notebook.add(tabCube, text="Cube")
notebook.add(tabCylinder, text="Cylinder")
notebook.add(tabSphere, text="Sphere")
notebook.pack(expand=True, fill="both")

menubar = Menu(window)
window.config(menu=menubar)

fileMenu = Menu(menubar, tearoff=0, font=("Arial", 10))
menubar.add_cascade(label="File", menu=fileMenu   #image=(PhotoImage),
compound='left  will display any photo image next to this option
                    )

fileMenu.add_separator()
```

```
fileMenu.add_command(label="Exit", command=quit)

editMenu = Menu(menubar, tearoff=0, font=("Arial", 10))
menubar.add_cascade(label="Edit", menu=editMenu)


settingsMenu = Menu(menubar, tearoff=0, font=("Arial", 10))
menubar.add_cascade(label="Settings", menu=settingsMenu)

#Label(tabCube, text="Hello, this is tab#1", width=50,
height=25,background="#404040").pack()
#Label(tabCylinder, text="Goodbye, this is tab#2", width=50,
height=25,background="#404040").pack()

photoimage = PhotoImage(file='../RTG_GUI/GUI/VXS-Space-Logo-For-GUI.png')







###################################################CUBE#####################
############################








ANSlabel = Label(tabCube, font=("Arial", 50, 'bold'), width=30, height=1,
background="#AFABAB")
ANSlabel.place(x=7, y=15)


VXS = Label(tabCube,
             text="Hello",
             fg='#262626',
```

```
             bg='#262626',
             image=photoimage)
VXS.place(x=10, y=425)


alabel = Label(tabCube,
             text="A: ",
             font=('Arial', 20, 'bold'),
             fg='white',
             bg='#262626',)

alabel.place(x=10,y=115)

Alabel = Label(tabCube, font=("Arial", 20, 'bold'), text="A is the side
length of the cube!", width=25, height=1, bg="#262626", fg="White")
Alabel.place(x=5, y=160)

entrya= Entry(tabCube,
             font=("Arial", 20, 'bold'),
             fg="black",
             bg="#AFABAB")


entrya.place(x=50, y=115)







##################################################CUBE######################
############################
```

```
#################################################CYLINDER#################################################




ANSlabelcy = Label(tabCylinder, font=("Arial", 50, 'bold'), width=30,
height=1, background="#AFABAB")
ANSlabelcy.place(x=7, y=15)


VXScy = Label(tabCylinder,
             text="Hello",
             fg='#262626',
             bg='#262626',
             image=photoimage)
VXScy.place(x=10, y=425)


rlabel = Label(tabCylinder,
             text="R: ",
             font=('Arial', 20, 'bold'),
             fg='white',
             bg='#262626',)

rlabel.place(x=10,y=115)

rlabel = Label(tabCylinder, font=("Arial", 20, 'bold'), text="R is the
Radius of the cylinder", width=25, height=1, bg="#262626", fg="White")
rlabel.place(x=-5, y=160)


entryr= Entry(tabCylinder,
             font=("Arial", 20, 'bold'),
             fg="black",
```

```
             bg="#AFABAB")


entryr.place(x=50, y=115)


helabel = Label(tabCylinder,
             text="H: ",
             font=('Arial', 20, 'bold'),
             fg='white',
             bg='#262626',)

helabel.place(x=10,y=205)

hlabel = Label(tabCylinder, font=("Arial", 20, 'bold'), text="H is the
height of the Cylinder", width=25, height=1, bg="#262626", fg="White")
hlabel.place(x=-5, y=250)

entryh= Entry(tabCylinder,
             font=("Arial", 20, 'bold'),
             fg="black",
             bg="#AFABAB")


entryh.place(x=50, y=205)

sumbitCY_button = Button(tabCylinder, text="Submit", font=("Arial", 10,
'bold'), command=submitcy, height=1, width=10, bg="#AFABAB", fg="Black")
sumbitCY_button.place(x=500, y=505)


##################################################CUBE######################
############################
```

```
##################################################CUBE######################
############################




ANSlabels = Label(tabSphere, font=("Arial", 50, 'bold'), width=30, height=1,
background="#AFABAB")
ANSlabels.place(x=7, y=15)


VXS = Label(tabSphere,
             text="Hello",
             fg='#262626',
             bg='#262626',
             image=photoimage)
VXS.place(x=10, y=425)


ralabel = Label(tabSphere,
             text="R: ",
             font=('Arial', 20, 'bold'),
             fg='white',
             bg='#262626',)

ralabel.place(x=10,y=115)

Ralabel = Label(tabSphere, font=("Arial", 20, 'bold'), text="R is the radius
of your sphere!", width=25, height=1, bg="#262626", fg="White")
Ralabel.place(x=-5, y=160)

entryra= Entry(tabSphere,
             font=("Arial", 20, 'bold'),
             fg="black",
             bg="#AFABAB")
```

```
entryra.place(x=50, y=115)


sumbits_button = Button(tabSphere, text="Submit", font=("Arial", 10,
'bold'), command=submits, height=1, width=10, bg="#AFABAB", fg="Black")
sumbits_button.place(x=500, y=505)





##################################################CUBE#####################
############################




# Define a list of available isotopes
isotopes = ["Pu-238", "Sr-90"]

sumbitQ_button = Button(tabCube, text="Submit", font=("Arial", 10, 'bold'),
command=submitk, height=1, width=10, bg="#AFABAB", fg="Black")
sumbitQ_button.place(x=500, y=505)

# Create the dropdown menu
isotope_var = StringVar(window)
isotope_var.set(isotopes[0])  # Set default selection

isotope_dropdown = ttk.Combobox(tabCube, textvariable=isotope_var,
values=isotopes)
isotope_dropdown.place(x=125, y=270)

# Label for the dropdown
isotope_label = Label(tabCube, text="Isotope:", font=('Arial', 20, 'bold'),
fg='white', bg='#262626')
isotope_label.place(x=10, y=260)
```

```
# Entry field for initial mass
entry_mass = Entry(tabCube, font=("Arial", 20, 'bold'), fg="black",
bg="#AFABAB", state="normal")
entry_mass.place(x=135, y=310)


# Label for initial mass
mass_label = Label(tabCube, text="Mass (g):", font=('Arial', 20, 'bold'),
fg='white', bg='#262626')
mass_label.place(x=10, y=310)


# Entry field for time elapsed
entry_time = Entry(tabCube, font=("Arial", 20, 'bold'), fg="black",
bg="#AFABAB", validate="key")
entry_time.place(x=300, y=360)

# Label for time elapsed
time_label = Label(tabCube, text="Time Elapsed (years):", font=('Arial', 20,
'bold'), fg='white', bg='#262626')
time_label.place(x=10, y=360)









isotope_dropdown2 = ttk.Combobox(tabCylinder, textvariable=isotope_var,
values=isotopes)
isotope_dropdown2.place(x=125, y=310)

# Label for the dropdown
isotope_label2 = Label(tabCylinder, text="Isotope:", font=('Arial', 20,
'bold'), fg='white', bg='#262626')
isotope_label2.place(x=10, y=300)
```

```
# Entry field for initial mass
entry_mass2 = Entry(tabCylinder, font=("Arial", 20, 'bold'), fg="black",
bg="#AFABAB", state="normal")
entry_mass2.place(x=135, y=350)


# Label for initial mass
mass_label2 = Label(tabCylinder, text="Mass (g):", font=('Arial', 20,
'bold'), fg='white', bg='#262626')
mass_label2.place(x=10, y=350)


# Entry field for time elapsed
entry_time2 = Entry(tabCylinder, font=("Arial", 20, 'bold'), fg="black",
bg="#AFABAB", validate="key")
entry_time2.place(x=300, y=395)

# Label for time elapsed
time_label2 = Label(tabCylinder, text="Time Elapsed (years):",
font=('Arial', 20, 'bold'), fg='white', bg='#262626')
time_label2.place(x=10, y=395)






isotope_dropdown3 = ttk.Combobox(tabSphere, textvariable=isotope_var,
values=isotopes)
isotope_dropdown3.place(x=50, y=260)

# Label for the dropdown
isotope_label3 = Label(tabSphere, text="Isotope:", font=('Arial', 20,
'bold'), fg='white', bg='#262626')
isotope_label3.place(x=10, y=260)

# Entry field for initial mass
entry_mass3 = Entry(tabSphere, font=("Arial", 20, 'bold'), fg="black",
bg="#AFABAB", state="normal")
```

```
entry_mass3.place(x=135, y=310)


# Label for initial mass
mass_label3 = Label(tabSphere, text="Mass (g):", font=('Arial', 20, 'bold'),
fg='white', bg='#262626')
mass_label3.place(x=10, y=310)


# Entry field for time elapsed
entry_time3 = Entry(tabSphere, font=("Arial", 20, 'bold'), fg="black",
bg="#AFABAB", validate="key")
entry_time3.place(x=300, y=360)

# Label for time elapsed
time_label3 = Label(tabSphere, text="Time Elapsed (years):", font=('Arial',
20, 'bold'), fg='white', bg='#262626')
time_label3.place(x=10, y=360)

thermoelectric_var = StringVar()
thermoelectric_dropdown = ttk.Combobox(tabCube,
textvariable=thermoelectric_var, state="readonly")
thermoelectric_dropdown["values"] = list(thermoelectric_data.keys())
thermoelectric_dropdown.pack(padx=10, pady=10)  # Adjust packing as needed
thermoelectric_dropdown.current(0)  # Set default selection (optional)

thermoelectric_dropdown.place(x=350, y=422)

thermoeletricLabel = Label(tabCube, text="Thermoeletric Material:",
font=('Arial', 20, 'bold'), fg='white', bg='#262626')
thermoeletricLabel.place(x=10, y=410)

load_resistance_label = Label(tabCube, text="Load Resistance (Ω):",
font=('Arial', 20, 'bold'), fg='white', bg='#262626')
load_resistance_label.place(x=650, y=115)  # Adjust coordinates as needed

# Entry field for load resistance
entry_resistance = Entry(tabCube, font=("Arial", 20, 'bold'), fg="black",
bg="#AFABAB")
```

```
entry_resistance.place(x=930, y=115)  # Adjust coordinates as needed

"""

class RTGCalculator:
   def __init__(self, initial_power, decay_constant, load_resistance):
       # Initialize RTG calculator with input parameters
       self.initial_power = initial_power
       self.decay_constant = decay_constant
       self.load_resistance = load_resistance

   def calculate_power_output(self):        # Calculate voltage using
Seebeck coefficient (α) and ΔT
       alpha = 0.200e-3  # Example Seebeck coefficient (adjust based on your
material)
       delta_T = 873  # Example temperature difference (in kelvin)

       voltage = alpha * delta_T

       # Power output using Joule's Law
       power_output = (voltage ** 2) / self.load_resistance

       return power_output

def test_rtg_calculator():
   # Sample input parameters
   initial_power = 100  # Example initial power (in Watts)
   decay_constant = 0.018  # Example decay constant
   load_resistance = 10  # Example load resistance (in Ohms)
#
#
#    # Create an instance of RTGCalculator
#    rtg_calculator = RTGCalculator(initial_power, decay_constant,
load_resistance)
#
#    # Test the calculate_power_output method
#   power_output = rtg_calculator.calculate_power_output()
#    print("Power Output:", power_output)
```

```
# Run the test
test_rtg_calculator()

"""

window.mainloop()
```