```
# Power City Simulation

# --- Initialization ---

# Define constants for default capacities (kW for power, kWh for energy)
CONSTANT DEFAULT_HYDRO_CAPACITY = 1000
CONSTANT DEFAULT_SOLAR_CAPACITY = 1000
CONSTANT DEFAULT_WIND_CAPACITY = 1000
CONSTANT DEFAULT_BATTERY_CAPACITY = 1000
CONSTANT DEFAULT_BATTERY_LEVEL = 500

# Define constants for battery and grid
CONSTANT DEFAULT_BATTERY_HEALTH = 100  # Percentage (0-100)
CONSTANT GRID_COST_PER_KWH = 0.15  # Cost in GBP

# Initialize capacities (read from config.py if available, else use defaults)
FUNCTION initialize_capacities():
  # INPUT: None
  # OUTPUT: Tuple (hydro_capacity, solar_capacity, wind_capacity,
  #            battery_capacity, battery_level)
  #  - hydro_capacity (numeric): Maximum power output of hydro source (kW).
  #  - solar_capacity (numeric): Maximum power output of solar source (kW).
  #  - wind_capacity (numeric): Maximum power output of wind source (kW).
  #  - battery_capacity (numeric): Maximum energy storage of battery (kWh).
  #  - battery_level (numeric): Initial energy stored in the battery (kWh).
```

```
  IF "config.py" exists THEN:
    TRY:
      Import capacities from config.py
    CATCH ImportError:
      Log warning: "Error importing config.py, using defaults."
      Set capacities to default values
  ELSE:
    Log warning: "config.py not found, using defaults."
    Set capacities to default values
  ENDIF

  RETURN (hydro_capacity, solar_capacity, wind_capacity,
        battery_capacity, battery_level)

# Call the function to get initialized capacities
(hydro_capacity, solar_capacity, wind_capacity, battery_capacity,
 battery_level) = initialize_capacities()

# Initialize active states for power sources
hydro_active = TRUE
solar_active = TRUE
wind_active = TRUE
battery_active = TRUE

# Initialize grid usage and cost
grid_usage = 0.0
```

```
grid_usage_cost = 0.0

total_savings = 0.0


# Initialize battery health and age

battery_health = DEFAULT_BATTERY_HEALTH

battery_age = 0  # Years

charge_cycles = 0


# --- Simulation Functions ---


# Simulate weather conditions

FUNCTION simulate_weather():

 # INPUT: None

 # OUTPUT: Tuple (temperature, wind_speed, solar_radiation)

 #  - temperature (numeric): Current temperature in degrees Celsius (°C).

 #  - wind_speed (numeric): Current wind speed in meters per second (m/s).

 #  - solar_radiation (numeric): Current solar radiation in watts per

 #                          square meter (W/m^2).


 Get current date and time


 # Simulate temperature with seasonal variation

 IF month is December, January, or February THEN: # Winter

  temperature = random value between -5 and 10 °C

 ELSE IF month is March, April, or May THEN: # Spring

  temperature = random value between 5 and 20 °C
```

```
  ELSE IF month is June, July, or August THEN: # Summer
    temperature = random value between 15 and 35 °C
  ELSE: # Autumn
    temperature = random value between 5 and 25 °C
  ENDIF

  # Simulate wind speed (higher during daytime)
  IF hour is between 6 AM and 6 PM THEN:
    wind_speed = random value between 0 and 20 m/s
  ELSE:
    wind_speed = random value between 0 and 10 m/s
  ENDIF

  # Simulate solar radiation (daytime only)
  IF hour is between 6 AM and 6 PM THEN:
    solar_radiation = random value between 0 and 1000 W/m^2
  ELSE:
    solar_radiation = 0
  ENDIF

  Log debug message with simulated weather data

  RETURN (temperature, wind_speed, solar_radiation)


# Simulate energy usage based on temperature
```

```
FUNCTION simulate_energy_usage(temperature):
 # INPUT:
 #  - temperature (numeric): Current temperature in degrees Celsius (°C).
 # OUTPUT:
 #  - energy_usage (numeric): Simulated energy consumption in kilowatt-hours (kWh).

 base_usage = 1000 kWh
 usage_variation = random value between -200 and 200 kWh

 # Increase usage for extreme temperatures
 IF temperature < 0°C OR temperature > 30°C THEN:
  Increase usage_variation by 200 kWh
 ENDIF

 energy_usage = base_usage + usage_variation

 Log debug message with simulated energy usage

 RETURN energy_usage


# Simulate hydroelectricity generation
FUNCTION simulate_hydroelectricity():
 # INPUT: None (uses global hydro_active and hydro_capacity)
 # OUTPUT:
 #  - hydro_power (numeric): Power generated by the hydro source in kilowatts (kW).
```

```
  IF hydro_active is FALSE THEN:
    RETURN 0
  ENDIF

  water_flow = random value between 50 and 500 m^3/s
  efficiency = 0.9
  hydro_power = (water_flow * efficiency * 9.81 * hydro_capacity) kW
  hydro_power = minimum of (hydro_power, hydro_capacity) # Limit to capacity

  Log debug message with simulated hydro power

  RETURN hydro_power


# Simulate solar power generation
FUNCTION simulate_solar_power(solar_radiation):
  # INPUT:
  #  - solar_radiation (numeric): Current solar radiation in watts per
  #                           square meter (W/m^2).
  # OUTPUT:
  #  - solar_power (numeric): Power generated by the solar source in
  #                        kilowatts (kW).

  IF solar_active is FALSE THEN:
    RETURN 0
```

```
  ENDIF

  efficiency = 0.2
  solar_power = (solar_radiation * efficiency * solar_capacity / 1000) kW
  solar_power = minimum of (solar_power, solar_capacity) # Limit to capacity

  Log debug message with simulated solar power

  RETURN solar_power


# Simulate wind power generation
FUNCTION simulate_wind_power(wind_speed):
  # INPUT:
  #  - wind_speed (numeric): Current wind speed in meters per second (m/s).
  # OUTPUT:
  #  - wind_power (numeric): Power generated by the wind source in
  #                    kilowatts (kW).

  IF wind_active is FALSE THEN:
    RETURN 0
  ENDIF

  efficiency = 0.4
  wind_power = (wind_speed^3 * efficiency * wind_capacity / 1000) kW
  wind_power = minimum of (wind_power, wind_capacity) # Limit to capacity
```

```
  Log debug message with simulated wind power

  RETURN wind_power

# --- Core Simulation Logic ---

# Update simulation data and variables
FUNCTION update_data():
  # INPUT: None (uses and updates global variables)
  # OUTPUT: None (updates global variables)
  # This function updates the simulation data, including weather, energy usage,
  # generation, battery level, grid usage, battery health, and cost calculations.

  Get current time

  # Simulate weather, energy usage, and power generation
  (temperature, wind_speed, solar_radiation) = simulate_weather()
  energy_usage = simulate_energy_usage(temperature)
  hydro_power = simulate_hydroelectricity()
  solar_power = simulate_solar_power(solar_radiation)
  wind_power = simulate_wind_power(wind_speed)

  total_generation = hydro_power + solar_power + wind_power
  net_energy = total_generation - energy_usage
```

```
# Battery and Grid Logic
IF battery_active is TRUE THEN:
 IF net_energy > 0 AND battery_level < battery_capacity THEN:
  # Charge the battery (consider battery health)
  battery_level = minimum of (battery_capacity,
                 battery_level + net_energy * (battery_health / 100))
  grid_usage = 0.0
  IF battery_level equals battery_capacity THEN:
   Increment charge_cycles
  ENDIF
 ELSE IF net_energy < 0 THEN:
  # Discharge the battery (consider battery health)
  battery_level = maximum of (0,
                 battery_level + net_energy * (battery_health / 100))
  IF battery_level equals 0 THEN:
   grid_usage = absolute value of (net_energy)
   grid_usage_cost = grid_usage_cost + (grid_usage * GRID_COST_PER_KWH)
  ELSE:
   grid_usage = 0.0
  ENDIF
 ENDIF
ELSE: # Battery is inactive
 IF net_energy < 0 THEN:
  grid_usage = absolute value of (net_energy)
  grid_usage_cost = grid_usage_cost + (grid_usage * GRID_COST_PER_KWH)
 ELSE:
```

```
    grid_usage = 0.0
  ENDIF
ENDIF

# Calculate savings
total_savings = (energy_usage - grid_usage) * GRID_COST_PER_KWH

# Simulate battery health degradation (hourly)
battery_age = battery_age + (1 / (365 * 24)) # Increment age by one hour
degradation_rate = 100 / (2 * 365 * 24) # 0% in 2 years
battery_health = maximum of (0, battery_health - degradation_rate)

# Store data
Add weather data to weather_data list: (current_time, temperature,
                                       wind_speed, solar_radiation)
Add energy data to energy_data list: (current_time, energy_usage,
                                     hydro_power, solar_power, wind_power,
                                     battery_level, grid_usage, battery_health,
                                     grid_usage_cost, total_savings)

# Limit data storage to the last 100 entries
IF length of weather_data > 100 THEN:
  Remove the first element from weather_data
  Remove the first element from energy_data
ENDIF
```

```
  Log debug message with all updated data

  # Update GUI if it is initialized
  IF "root" is defined THEN:
    Call root.after(1000, update_text_field, current_time, temperature, ...)
  ENDIF

# --- GUI Functions ---

# Update the text field in the GUI with simulation data
FUNCTION update_text_field(current_time, temperature, wind_speed,
                    solar_radiation, energy_usage, hydro_power,
                    solar_power, wind_power, battery_level,
                    grid_usage, battery_health, grid_usage_cost,
                    total_savings):
  # INPUT: All the current simulation data (time, weather, energy, etc.)
  # OUTPUT: None (updates the GUI text field)

  Set text_field state to NORMAL (editable)
  Clear the text_field
  Insert all the simulation data into the text_field with appropriate formatting
  Set text_field state to DISABLED (read-only)
  Call update_weather_readings(temperature, wind_speed, solar_radiation)
  Call update_battery_readings(battery_level, battery_health, battery_age,
                      charge_cycles, grid_usage_cost, total_savings)
```

```
# Update weather readings labels in the GUI
FUNCTION update_weather_readings(temperature, wind_speed, solar_radiation):
  # INPUT: Current weather data
  # OUTPUT: None (updates GUI labels)
  Update temperature_label with temperature value
  Update wind_speed_label with wind_speed value
  Update solar_radiation_label with solar_radiation value

# Update battery readings labels in the GUI
FUNCTION update_battery_readings(battery_level, battery_health, battery_age,
                                 charge_cycles, grid_usage_cost, total_savings):
  # INPUT: Current battery and cost data
  # OUTPUT: None (updates GUI labels)
  Update battery_level_label with battery_level value
  Update battery_health_label with battery_health value
  Update battery_age_label with battery_age value
  Update charge_cycles_label with charge_cycles value
  Update grid_usage_cost_label with grid_usage_cost value
  Update total_savings_label with total_savings value


# --- Graphing Functions ---

# Update and animate the matplotlib graphs
FUNCTION animate(_):
  # INPUT: _ (placeholder argument for animation function)
```

```
  # OUTPUT: None (updates the matplotlib graphs)

  Call update_data() to get the latest simulation data

  Create a pandas DataFrame from energy_data

  Clear all subplots (ax1, ax2, ax3, ax4, ax5)

  # Plot data on each subplot (hydro, solar, wind, battery, grid)
  # ... (Code for plotting data on each subplot with labels, titles, etc.)

  Adjust layout for better readability

# --- GUI Setup ---

# Create the main window
root = tk.Tk()
Set window title to "Power City Simulation"
Set window geometry to "1400x900"

# Configure grid layout for the main window
# ...

# Create canvas, scrollbars, and frame for the main content
# ...
```

```
# Create frame for matplotlib figure
# ...

# Create canvas for matplotlib figure and pack it
# ...

# Create frame for buttons (Start, Stop)
# ...

# Create frame for renewable source controls (Hydro, Solar, Wind, Battery)
# ...

# Create frame for capacity input fields (Hydro, Solar, Wind, Battery)
# ...

# Create frame for text output of simulation data
# ...

# Create frame for weather readings labels
# ...

# Create frame for battery readings labels
# ...

# Create frame for grid cost breakdown labels
# ...
```

```
# --- Button Functions ---

# Start the graph animation
FUNCTION start_animation():
  Start the animation
  Play sound notification

# Stop the graph animation
FUNCTION stop_animation():
  Stop the animation
  Play sound notification

# Toggle hydro power source
FUNCTION toggle_hydro():
  Toggle hydro_active state
  Update hydro_button text and color
  Log debug message
  Play sound notification

# Toggle solar power source
FUNCTION toggle_solar():
  Toggle solar_active state
  Update solar_button text and color
  Log debug message
  Play sound notification
```

```
# Toggle wind power source
FUNCTION toggle_wind():
  Toggle wind_active state
  Update wind_button text and color
  Log debug message
  Play sound notification

# Toggle battery usage
FUNCTION toggle_battery():
  Toggle battery_active state
  Update battery_button text and color
  Log debug message
  Play sound notification

# Update hydro status (used for initial setup)
FUNCTION update_hydro_status(status):
  Set hydro_active to status
  Update hydro_button text and color
  Log debug message

# Update solar status (used for initial setup)
FUNCTION update_solar_status(status):
  Set solar_active to status
  Update solar_button text and color
  Log debug message
```

```
# Update wind status (used for initial setup)
FUNCTION update_wind_status(status):
  Set wind_active to status
  Update wind_button text and color
  Log debug message

# Update battery status (used for initial setup)
FUNCTION update_battery_status(status):
  Set battery_active to status
  Update battery_button text and color
  Log debug message

# Play sound notification (cross-platform)
FUNCTION play_sound():
  IF platform is Windows THEN:
    Play Windows system sound
  ELSE IF platform is macOS THEN:
    Play macOS system sound
  ELSE: # Linux
    Play Linux system sound
  ENDIF

# --- Capacity and Grid Cost Functions ---

# Update capacity display labels in the GUI
```

```
FUNCTION update_capacity_display():
  Update hydro_capacity_label with hydro_capacity value
  Update solar_capacity_label with solar_capacity value
  Update wind_capacity_label with wind_capacity value
  Update battery_capacity_label with battery_capacity value

# Update energy source capacities based on user input
FUNCTION update_capacities():
  TRY:
    Get hydro_capacity from hydro_capacity_entry
    Get solar_capacity from solar_capacity_entry
    Get wind_capacity from wind_capacity_entry
    Get battery_capacity from battery_capacity_entry
    Ensure battery_level does not exceed battery_capacity
    Call update_capacity_display()
    Show success messagebox: "Capacities updated!"
    Call update_data() to reflect changes in the simulation
    Play sound notification
  CATCH ValueError:
    Log error message
    Show error messagebox: "Please enter valid numbers for capacities."
    Play sound notification

# Update grid cost per kWh based on user input
FUNCTION update_grid_cost():
  TRY:
```

```
    Get GRID_COST_PER_KWH from grid_cost_entry
    Update grid_cost_label with new GRID_COST_PER_KWH value
    Show success messagebox: "Grid cost updated!"
    Play sound notification
  CATCH ValueError:
    Log error message
    Show error messagebox: "Please enter a valid number for grid cost."
    Play sound notification

# Update grid cost display label in the GUI
FUNCTION update_grid_cost_display():
  Update grid__cost_label with GRID_COST_PER_KWH value

# --- Report Generation Functions ---

# Generate a report string from simulation data
FUNCTION generate_report(data, title):
  # INPUT:
  #  - data (list): List of simulation data entries.
  #  - title (string): Title of the report (e.g., "Daily Report").
  # OUTPUT:
  #  - report_text (string): Formatted report string containing the simulation data.

  Initialize report_text with the title and current capacities

  FOR EACH entry in data:
```

```
    Extract data values (time, energy usage, power generation, etc.)
    Add formatted data to report_text
  ENDFOR

  RETURN report_text

# Update the report text field in the GUI
FUNCTION update_report_text(report_text):
  # INPUT:
  #  - report_text (string): The formatted report text to display.
  # OUTPUT: None (updates the GUI report text field)

  Set report_text_field state to NORMAL (editable)
  Clear the report_text_field
  Insert report_text into report_text_field
  Set report_text_field state to DISABLED (read-only)

# Generate and display the daily report
FUNCTION generate_daily_report():
  Get current time
  Calculate time 24 hours ago

  Filter energy_data to get entries from the last 24 hours

  IF no daily data THEN:
    Show messagebox: "No data for the last 24 hours."
```

```
  Play sound notification
  RETURN
ENDIF

# Group data by hour
Create an empty dictionary hourly_data

FOR EACH entry in daily_data:
  Extract the hour from the entry's timestamp
  IF hour is not in hourly_data THEN:
    Add hour as key to hourly_data with an empty list as its value
  ENDIF
  Append the entry to the list associated with the hour in hourly_data
ENDFOR

Initialize report_text with "Daily Report" and current capacities

Initialize total_cost_savings to 0.0

FOR EACH hour, entries in sorted hourly_data:
  Calculate average energy usage, grid usage, and costs for the hour
  Add formatted hourly data to report_text
  Accumulate total_cost_savings
ENDFOR

Add total_cost_savings to report_text
```

```
  Call update_report_text(report_text) to display the report
  Play sound notification

# Generate and display the weekly report (similar logic to daily report)
FUNCTION generate_weekly_report():
  # ... (Implementation for weekly report generation)

# Generate and display the monthly report (similar logic to daily report)
FUNCTION generate_monthly_report():
  # ... (Implementation for monthly report generation)

# Generate and display the yearly report (similar logic to daily report)
FUNCTION generate_yearly_report():
  # ... (Implementation for yearly report generation)

# --- Main Program Execution ---

# Set up the matplotlib figure and axes
# ...

# Hide unused subplot
# ...

# Create animation object
# ...
```

```
# Adjust layout
# ...

# Call update_capacities() to display initial capacities
# ...

# Start the main Tkinter event loop
root.mainloop()
```