```python

# *** REFERENCES ***

# regarding the topic:
# Electric Machinery Fundamentals, Stephen J. Chapman, ed.5

# regarding the gui library and code:
# https://docs.pysimplegui.com/en/latest/
# https://youtu.be/kQ8DGP9p2LY?si=1KOmzYeDvAGHe7ox
# https://matplotlib.org/stable/users/index
# https://stackoverflow.com/



import PySimpleGUI as sg
import numpy as np
from math import floor, log10
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg



# ***BUILDING THE UI***


sg.theme('Default')
font = ('Aptos', 13)

app_title = 'SINGLE PHASE TRANSFORMER CALCULATOR'
course = 'EEE-3003: Electromechanical Energy Conversion'
author1 = 'Ahmad Zameer NAZARI'
author2 = 'Ismet Mert ŞEN'


# INPUT TAB

tab_input_instr = 'Provide all given values to obtain transformer characteristics'
tab_input_note = '*Note: only step-down voltages permitted'

col_input_data = sg.Column([
    [sg.Push(), sg.Text('Power Rating: '),sg.Input(15000,key = '-INP_S-', size=7), sg.Text('VA', s=5)],
    [sg.Push(), sg.Text('Primary Voltage: '), sg.Input(2300,key = '-INP_V_P-', size=7), sg.Text('V', s=5)],
    [sg.Push(), sg.Text('Secondary Voltage: '), sg.Input(230,key = '-INP_V_S-', size=7), sg.Text('V', s=5)]
])

col_input_oc = sg.Column([
```

```
    [sg.Push(), sg.Image('images/V_OC.png'), sg.Input(230,key = '-INP_V_OC-', size=5), sg.Text('V', s=5)],
    [sg.Push(), sg.Image('images/I_OC.png'), sg.Input(2.1,key = '-INP_I_OC-', size=5), sg.Text('A', s=5)],
    [sg.Push(), sg.Image('images/P_OC.png'), sg.Input(50,key = '-INP_P_OC-', size=5), sg.Text('W', s=5)]
])

col_input_sc = sg.Column([
    [sg.Push(), sg.Image('images/V_SC.png'), sg.Input(47,key = '-INP_V_SC-', size=5), sg.Text('V', s=5)],
    [sg.Push(), sg.Image('images/I_SC.png'), sg.Input(6,key = '-INP_I_SC-', size=5), sg.Text('A', s=5)],
    [sg.Push(), sg.Image('images/P_SC.png'), sg.Input(160,key = '-INP_P_SC-', size=5), sg.Text('W', s=5)]
])

col_ratio = sg.Column([
    [
        sg.Text('Transformer Ratio,      '),
        sg.Image('images/N.png'),
        sg.Input(key = '-OUT_RATIO-', disabled=True, s=5),
        sg.Text('', s=5)
    ]
])

col_input = sg.Column([
    [sg.Sizer(5,5)],
    [col_input_data],
    [sg.Sizer(5,5)],
    [sg.Text('Open Circuit Test', expand_x=True, justification='center'), col_input_oc],
    [sg.Sizer(5,5)],
    [sg.Text('Short Circuit Test', expand_x=True, justification='center'), col_input_sc],
    [sg.Sizer(5,5)],
    [col_ratio],
    [sg.Sizer(5,5)]
], element_justification='right')

frame_input = sg.Column([
    [
        sg.Frame('',
                 layout = [
                     [sg.Sizer(15,15), col_input, sg.Sizer(10,10)]
                 ])
    ]
])


```

```python
# IMPEDANCE AND EQUIV CIRCUIT TAB

col_imp_refer_p_ser = sg.Column([
    [
        sg.Image('images/Z_eq.png', s=(50,20)),
        sg.Multiline(key = '-OUT_Z_EQ_P-', s=(15,2), disabled=True, background_color='#F0F0F0', no_scrollbar=True)
    ],
    [sg.Image('images/R_eq.png', s=(50,20)), sg.Input(key = '-OUT_R_EQ_P-', disabled=True, s=15)],
    [sg.Image('images/X_l_eq.png', s=(50,20)), sg.Input(key = '-OUT_X_L_EQ_P-', disabled=True, s=15)]
])

col_imp_refer_p_phi = sg.Column([
    [
        sg.Image('images/Y_phi.png', s=(50,20)),
        sg.Multiline(key = '-OUT_Y_PHI_P-', s=(15,2), disabled=True, background_color='#F0F0F0', no_scrollbar=True)
    ],
    [sg.Image('images/G_phi.png', s=(50,20)), sg.Input(key = '-OUT_G_PHI_P-', disabled=True, s=15)],
    [sg.Image('images/B_phi.png', s=(50,20)), sg.Input(key = '-OUT_B_PHI_P-', disabled=True, s=15)],
    [
        sg.Image('images/Z_phi.png', s=(50,20)),
        sg.Multiline(key = '-OUT_Z_PHI_P-', s=(15,2), disabled=True, background_color='#F0F0F0', no_scrollbar=True)
    ],
    [sg.Image('images/R_c.png', s=(50,20)), sg.Input(key = '-OUT_R_PHI_P-', disabled=True, s=15)],
    [sg.Image('images/X_m.png', s=(50,20)), sg.Input(key = '-OUT_X_PHI_P-', disabled=True, s=15)]
])

col_imp_refer_s_ser = sg.Column([
    [
        sg.Image('images/Z_eq.png', s=(50,20)),
        sg.Multiline(key = '-OUT_Z_EQ_S-', s=(15,2), disabled=True, background_color='#F0F0F0', no_scrollbar=True)
    ],
    [sg.Image('images/R_eq.png', s=(50,20)), sg.Input(key = '-OUT_R_EQ_S-', disabled=True, s=15)],
    [sg.Image('images/X_l_eq.png', s=(50,20)), sg.Input(key = '-OUT_X_L_EQ_S-', disabled=True, s=15)]
])

col_imp_refer_s_phi = sg.Column([
    [
        sg.Image('images/Y_phi.png', s=(50,20)),
```

```python
        sg.Multiline(key = '-OUT_Y_PHI_S-', s=(15,2), disabled=True, background_color='#F0F0F0', no_scrollbar=True)
    ],
    [sg.Image('images/G_phi.png', s=(50,20)), sg.Input(key = '-OUT_G_PHI_S-', disabled=True, s=15)],
    [sg.Image('images/B_phi.png', s=(50,20)), sg.Input(key = '-OUT_B_PHI_S-', disabled=True, s=15)],
    [
        sg.Image('images/Z_phi.png', s=(50,20)),
        sg.Multiline(key = '-OUT_Z_PHI_S-', s=(15,2), disabled=True, background_color='#F0F0F0', no_scrollbar=True)
    ],
    [sg.Image('images/R_c.png', s=(50,20)), sg.Input(key = '-OUT_R_PHI_S-', disabled=True, s=15)],
    [sg.Image('images/X_m.png', s=(50,20)), sg.Input(key = '-OUT_X_PHI_S-', disabled=True, s=15)]
])

frame_imp_refer_p = sg.Column([
    [
        sg.Frame('Referred to Primary',
              layout = [
                  [sg.Sizer(25,25)],
                  [col_imp_refer_p_ser, col_imp_refer_p_phi, sg.Sizer(10,10), sg.Image('images/Refer_p.png')],
                  [sg.Sizer(25,25)]
              ])
    ]
])

frame_imp_refer_s = sg.Column([
    [sg.Frame('Referred to Secondary',
              layout = [
                  [sg.Sizer(25,25)],
                  [sg.Image('images/Refer_s.png'), sg.Sizer(10,10), col_imp_refer_s_ser, col_imp_refer_s_phi],
                  [sg.Sizer(25,25)]
              ])]
])


# VOLTAGE REGULATION TAB

col_vr_calc = sg.Column([
    [
        sg.Text('Rated Secondary Current,'),
        sg.Image('images/I_2,rated.png', s=(50,20)),
        sg.Input(key = '-OUT_I_S_RATED-', disabled=True, s=10)
    ],
    [
```

```
        sg.Text('Full Load Secondary Voltage,'),
        sg.Image('images/V_2,fl.png', s=(50,20)),
        sg.Input(key = '-OUT_V_S_FL-', disabled=True, s=10)
    ],
    [
        sg.Text('No Load Secondary Voltage,'),
        sg.Image('images/V_2,nl.png', s=(50,20)),
        sg.Input(key = '-OUT_V_S_NL-', disabled=True, s=10)
    ],
    [
        sg.Text('Voltage Regulation,'),
        sg.Image('images/VR.png', s=(50,20)),
        sg.Input(key = '-OUT_VR-', disabled=True, s=10)
    ],
    [
        sg.Text('Voltage Regulation at 0.8PF lagging (Inductive Load),'),
                sg.Image('images/VR_0.8,lag.png', s=(75,20)),
        sg.Input(key = '-OUT_VR_LAG-', disabled=True, s=10)
    ],
    [
        sg.Text('Voltage Regulation at 0.8PF leading (Capacitive Load),'),
                sg.Image('images/VR_0.8,lead.png', s=(75,20)),
        sg.Input(key = '-OUT_VR_LEAD-', disabled=True, s=10)
    ]
], element_justification='right')

frame_vr_calc = sg.Column([
    [sg.Frame('',
              layout=[
                  [sg.Sizer(10,10)],
                  [sg.Sizer(20,1), col_vr_calc, sg.Sizer(20,1)],
                  [sg.Sizer(10,10)]
              ])]
])

col_vr_canvas = sg.Column([
    [
        sg.Frame('Plot',
             layout = [
                        [sg.Sizer(15,15)],
                        [sg.Sizer(20,20), sg.Canvas(key = '-
VR_CANVAS-'),sg.Sizer(20,20)],
                        [sg.Sizer(20,20)]
                        ]
                    )
    ]
])


```

```
# EFFICIENCY TAB

col_eff_calc = sg.Column([
    [   sg.Text('Input Power,'),
        sg.Image('images/P_in.png', s=(40,20)),
        sg.Input(key = '-OUT_P_IN-', disabled=True, s=10)
    ],
    [
        sg.Text('Output Power,'),
        sg.Image('images/P_out.png', s=(40,20)),
        sg.Input(key = '-OUT_P_OUT-', disabled=True, s=10)
    ],
    [
        sg.Text('Copper Loss,'),
        sg.Image('images/P_Cu.png', s=(40,20)),
        sg.Input(key = '-OUT_P_CU-', disabled=True, s=10)
    ],
    [
        sg.Text('Core Loss,'),
        sg.Image('images/P_core.png',s=(40,20)),
        sg.Input(key = '-OUT_P_C-', disabled=True, s=10)
    ],
    [
        sg.Text('Efficiency,'),
        sg.Image('images/eta.png',s=(40,20)),
        sg.Input(key = '-OUT_EFF-', disabled=True, s=10)
    ]
], element_justification='right')

frame_eff_calc = sg.Column([
    [sg.Frame('',
              layout=[
                  [sg.Sizer(10,10)],
                  [sg.Sizer(70,1), col_eff_calc, sg.Sizer(70,1)],
                  [sg.Sizer(10,10)]
              ])]
])

col_eff_canvas = sg.Column([
    [
        sg.Frame('Plot',
             layout = [
                    [sg.Sizer(15,15)],
                    [sg.Sizer(20,150), sg.Canvas(key = '-EFF_CANVAS-'),sg.Sizer(20,150)],
                    [sg.Sizer(20,20)]
                 ]
                 )
    ]
```

```python
])


col_crd = [
    [sg.Text(course, font=('JetBrains Mono', 15, 'bold'))],
    [sg.Sizer(30,30)],
    [sg.Text(author1, font=('JetBrains Mono', 12))],
    [sg.Sizer(5,5)],
    [sg.Text(author2, font=('JetBrains Mono', 12))],
    [sg.Sizer(30,30)]
]


# LAYING DOWN ALL TABS TOGETHER IN THE WINDOW

tab_input = [
            [sg.Sizer(25,25)],
            [sg.Text(app_title, font=('Aptos', 15, 'bold'), s=3, justification='center', expand_x=True)],
            [sg.Sizer(15,15)],
            [sg.Text(tab_input_instr,justification='center', expand_x=True)],
            [sg.Text(tab_input_note, font=('Aptos', 9), colors='red',justification='center', expand_x=True)],
            [sg.Sizer(5,5)],
            [sg.Push(), frame_input, sg.Image('images/Transformer.png') , sg.Push()],
            [sg.Sizer(25,25)],
            [sg.Push(), sg.Button('Calculate', key='-CALCULATE-', enable_events=True, expand_x=True), sg.Push()],
            [sg.Sizer(5,5)],
            [sg.Push(), sg.Text(key='-WARNING-', font=('Aptos', 10), colors='red', justification='center', expand_x=True), sg.Push()]
           ]
tab_imp = [
            [sg.Sizer(25,25)],
            [sg.Push(), frame_imp_refer_p, sg.Push()],
            [sg.Sizer(25,25)],
            [sg.Push(), frame_imp_refer_s, sg.Push()],
            [sg.Sizer(25,25)]
           ]
tab_vr = [[sg.Push(), frame_vr_calc, sg.Push()], [sg.Push(), col_vr_canvas, sg.Push()]]
tab_eff =  [[sg.Push(), frame_eff_calc, sg.Push()], [sg.Sizer(28,28)], [sg.Push(), col_eff_canvas, sg.Push()]]
tab_crd = [[sg.VPush()], [sg.Push(), sg.Column(col_crd, element_justification='c'), sg.Push()], [sg.VPush()]]

layout = [[sg.TabGroup(
    [[
        sg.Tab('Input', tab_input),
        sg.Tab('Impedances', tab_imp),
```

```
        sg.Tab('Voltage Regulation', tab_vr),
        sg.Tab('Efficiency', tab_eff),
        sg.Tab('*', tab_crd)
    ]]
)]]

window = sg.Window(app_title.title(), layout, font=font, finalize=True, element_justification='c')




# ***CALCLATIONS***

# more accurate significant figure function for smaller values
def sig_figs(x: float, precision: int):
    x = float(x)
    precision = int(precision)

    return round(x, -int(floor(log10(abs(x)))) + (precision - 1))


# transformer ratio

def calc_ratio(v_p, v_s):
    v_p = int(v_p)
    v_s = int(v_s)

    t_ratio = int(v_p/v_s)

    return t_ratio


# series equivalent impedance function

def calc_z_eq(v_sc, i_sc, p_sc):
    v_sc = float(v_sc)
    i_sc = float(i_sc)
    p_sc = float(p_sc)


    pf = p_sc / (v_sc * i_sc)

    # try:
    #     isinstance(pf, complex)
    # except:
    #     print('Entered short circuit parameters return complex power factor!')
    #     return

```

```
    z_eq_mag = round(v_sc/i_sc,2)
    z_eq_ang = round((np.arccos(pf)*180/np.pi),2)

    r_eq = round(p_sc/(i_sc**2),2)
    x_l_eq = round(np.sqrt((z_eq_mag**2) - (r_eq**2)),2)

    return r_eq, x_l_eq, z_eq_mag, z_eq_ang


# parallel (excitation branch) impedance function

def calc_z_phi(v_oc, i_oc, p_oc):
    v_oc = float(v_oc)
    i_oc = float(i_oc)
    p_oc = float(p_oc)

    pf = round(p_oc / (v_oc * i_oc),3)

    y_phi_mag = sig_figs((i_oc/v_oc),3)
    y_phi_ang = sig_figs((-np.arccos(pf)*180/np.pi),3)

    g_phi = sig_figs(y_phi_mag * pf, 3)
    b_phi = sig_figs(y_phi_mag * np.sin(y_phi_ang * np.pi / 180),3)

    r_phi = sig_figs((1 / g_phi), 3)
    x_phi = sig_figs(1 / abs(b_phi), 3)
    z_phi_mag = sig_figs((1 / y_phi_mag), 3)
    z_phi_ang = -y_phi_ang

    return g_phi, b_phi, y_phi_mag, y_phi_ang, r_phi, x_phi, z_phi_mag, z_phi_ang


# find values referred to primary or secondary

def refer_to_s(value_at_p, t_ratio):

    value_at_p = float(value_at_p)
    t_ratio = int(t_ratio)

    value_at_s = sig_figs((value_at_p/(t_ratio**2)), 3)

    return value_at_s

def refer_to_p(value_at_s, t_ratio):

    value_at_s = float(value_at_s)
    t_ratio = int(t_ratio)

    value_at_p = sig_figs((value_at_s*(t_ratio**2)), 3)
```

```python

    return value_at_p


# voltage regulation

def calc_vr(s_rated, v_p, v_s, r_eq, x_l_eq):

    s_rated = float(s_rated)
    v_p = float(v_p)
    v_s = float(v_s)
    r_eq = float(r_eq/100)
    x_l_eq = float(x_l_eq/100)

    i_s = round((s_rated / v_s), 2)

    v_p_a = complex(v_s,0) + (r_eq * complex(i_s,0)) + (complex(0,x_l_eq)*complex(i_s,0))
    # v_p_a = v_s + (r_eq * i_s) + ((x_l_eq*1j)*i_s)
    v_s_nl = round(abs(v_p_a),2)

    vr = sig_figs(((v_s_nl - v_s) / v_s) * 100,2)

    pf_lag = 0.8
    pf_lead = 0.8
    i_s_ang_lag = (-np.arccos(pf_lag))
    i_s_ang_lead = (np.arccos(pf_lead))
    i_s_lag = complex(i_s*np.cos(i_s_ang_lag), i_s*np.sin(i_s_ang_lag))
    i_s_lead = complex(i_s*np.cos(i_s_ang_lead), i_s*np.sin(i_s_ang_lead))

    v_p_a_lag = complex(v_s,0) + (r_eq * i_s_lag) + (complex(0,x_l_eq)*i_s_lag)
    v_p_a_lead = complex(v_s,0) + (r_eq * i_s_lead) + (complex(0,x_l_eq)*i_s_lead)
    v_s_nl_lag = round(abs(v_p_a_lag),2)
    v_s_nl_lead = round(abs(v_p_a_lead),2)

    vr_lag = sig_figs(((v_s_nl_lag - v_s) / v_s) * 100, 2)
    vr_lead = sig_figs(((v_s_nl_lead - v_s) / v_s) * 100, 2)



    i_s_range = np.linspace(0,i_s,100)
    i_s_lag_range = i_s_range*np.cos(i_s_ang_lag) + (i_s_range*np.sin(i_s_ang_lag)*1j)
    i_s_lead_range = i_s_range*np.cos(i_s_ang_lead) + (i_s_range*np.sin(i_s_ang_lead)*1j)

    v_p_a_range = v_s + (r_eq * i_s_range) + ((x_l_eq*1j)*i_s_range)
    v_p_a_lag_range = v_s + (r_eq * i_s_lag_range) + ((x_l_eq*1j)*i_s_lag_range)
    v_p_a_lead_range = v_s + (r_eq * i_s_lead_range) + ((x_l_eq*1j)*i_s_lead_range)
```

```python

    vr_range = ((np.absolute(v_p_a_range) - v_s) / v_s) * 100
    vr_lag_range = ((np.absolute(v_p_a_lag_range) - v_s) / v_s) * 100
    vr_lead_range = ((np.absolute(v_p_a_lead_range) - v_s) / v_s) * 100


    return i_s, v_s, v_s_nl, vr, vr_lag, vr_lead, \
        i_s_range, vr_range, i_s_lag_range, vr_lag_range, i_s_lead_range, vr_lead_range


# transformer efficiency function

def calc_eff(s_rated, v_s, i_s, v_s_nl, r_eq, r_phi):

    s_rated = float(s_rated)
    v_s = float(v_s)
    i_s = float(i_s)
    v_s_nl = float(v_s_nl)
    r_eq = float(r_eq/100)
    r_phi = float(r_phi)


    pow_out = round(v_s * i_s, 2)
    loss_cu = round((i_s**2)*r_eq, 2)
    loss_core = round((v_s_nl**2)/r_phi, 2) # voltage dependent. doesnt vary with load
    pow_in = round(pow_out + loss_cu + loss_core, 2)

    eff = round((pow_out / pow_in) * 100, 2)

    i_s = s_rated / v_s
    i_s_range_eff = np.linspace(0, 2*i_s,100)

    pow_out_range = v_s * i_s_range_eff

    loss_cu_range = (i_s_range_eff**2)*r_eq

    pow_in_range = pow_out_range + loss_cu_range + loss_core

    loss_cu_range_perc = loss_cu_range
    loss_core_range_perc = [loss_core] * len(i_s_range_eff)
    eff_range = (pow_out_range / pow_in_range ) * 100


    return pow_in, pow_out, loss_cu, loss_core, eff, \
        eff_range, i_s_range_eff, loss_cu_range_perc, loss_core_range_perc


```

```python

# *** PLOTS ***

# VR plot

fig_vr = plt.figure(1,figsize = (6,4.5))
fig_vr.add_subplot(111).plot([],[])
tkcanvas_agg_vr = FigureCanvasTkAgg(fig_vr, window['-VR_CANVAS-'].TKCanvas)
tkcanvas_agg_vr.draw()
tkcanvas_agg_vr.get_tk_widget().pack()

def vr_plot(i_s_range, vr_range,
                  i_s_lag_range, vr_lag_range,
                  i_s_lead_range, vr_lead_range):

    ax = fig_vr.axes
    ax[0].cla()
    ax[0].plot(i_s_range, vr_range)
    ax[0].plot(i_s_lag_range, vr_lag_range, 'r-.')
    ax[0].plot(i_s_lead_range, vr_lead_range, 'g--')
    ax[0].set_xlabel('Load (A)')
    ax[0].set_ylabel('VR (%)')
    ax[0].set_title('VR variation with load amount and type')
    ax[0].legend(['1 PF', '0.8 lag PF', '0.8 lead PF'], loc='best')
    tkcanvas_agg_vr.draw()
    tkcanvas_agg_vr.get_tk_widget().pack()


# efficiency plot

fig_eff = plt.figure(2,figsize = (6,4.5))
fig_eff.add_subplot(111).plot([],[])
tkcanvas_agg_eff = FigureCanvasTkAgg(fig_eff, window['-EFF_CANVAS-'].TKCanvas)
tkcanvas_agg_eff.draw()
tkcanvas_agg_eff.get_tk_widget().pack()

def eff_plot(i_s_range_eff, eff_range, loss_cu_range_perc, loss_core_range_perc):

    axes = fig_eff.axes
    axes[0].cla()
    axes[0].plot(i_s_range_eff, eff_range, label='Efficiency')
    axes[0].set_xlabel('Current Load (A)')
    axes[0].set_ylabel('Efficiency (%)')
    axes[0].set_ylim(80,100)

    for ax in fig_eff.axes:
        if ax is not axes[0]:
            fig_eff.delaxes(ax)

```

```
    ax1 = axes[0].twinx()

    ax1.plot(i_s_range_eff, loss_cu_range_perc,'r-.', label='Copper loss')
    ax1.plot(i_s_range_eff, loss_core_range_perc,'g--', label='Core loss')
    ax1.set_ylabel('Loss (W)')
    ax1.set_title('Efficiency variation with load')

    lines, labels = axes[0].get_legend_handles_labels()
    lines2, labels2 = ax1.get_legend_handles_labels()
    ax1.legend(lines + lines2, labels + labels2, loc='best')

    tkcanvas_agg_eff.draw()
    tkcanvas_agg_eff.get_tk_widget().pack()



# *** WHEN WINDOW ACTIVE ***

# for input validation

prompt = window['-WARNING-'].update
input_key_list = [key for key, value in window.key_dict.items()
    if isinstance(value, sg.Input)]
input_key_list_slice = input_key_list[:9]

while True:
    event, values = window.read()
    if event == sg.WIN_CLOSED:
        break


    # ON CALCULATE KEYPRESS

    if event == '-CALCULATE-':


        # validate input fields not empty

        if all(map(str.strip, [values[key] for key in input_key_list_slice])):
            # prompt("Calculated!")


            # validate input fields numbers

            if all(isinstance(values.get(key, ""), str) and any(char.isnumeric() for
char in values[key]) for key in input_key_list_slice):
                prompt('Calculated!')


```

```
                # call transformer ratio function, return its value
                t_ratio = calc_ratio(values['-INP_V_P-'], values['-INP_V_S-'])
                out_t_ratio = t_ratio
                window['-OUT_RATIO-'].update(out_t_ratio)


                # call series impedance function, return outputs then display
                # series impedance retrieved is that referred to primary
                # since LV secondary is short circuited and values are referred to primary
                # refer_to_s function is called to find values referred to secondary
                r_eq, x_l_eq, z_eq_mag, z_eq_ang = calc_z_eq(
                    values['-INP_V_SC-'], values['-INP_I_SC-'], values['-INP_P_SC-']
                    )

                z_eq_polar = f'{z_eq_mag} \u2220 {z_eq_ang} \u03a9'
                z_eq_rect = f'{r_eq}{x_l_eq:+}j \u03a9'

                out_z_eq = f'{z_eq_polar} \n {z_eq_rect}'
                out_r_eq = f'{r_eq} \u03a9'
                out_x_l_eq = f'{x_l_eq}j \u03a9'

                window['-OUT_Z_EQ_P-'].update(out_z_eq)
                window['-OUT_R_EQ_P-'].update(out_r_eq)
                window['-OUT_X_L_EQ_P-'].update(out_x_l_eq)

                z_eq_mag_refer_s = refer_to_s(z_eq_mag, t_ratio)
                r_eq_refer_s = refer_to_s(r_eq, t_ratio)
                x_l_eq_refer_s = refer_to_s(x_l_eq, t_ratio)

                z_eq_polar_refer_s = f'{z_eq_mag_refer_s}\u2220{z_eq_ang} \u03a9'
                z_eq_rect_refer_s = f'{r_eq_refer_s}{x_l_eq_refer_s:+}j \u03a9'

                out_z_eq_refer_s = f'{z_eq_polar_refer_s}\n{z_eq_rect_refer_s}'
                out_r_eq_refer_s = f'{r_eq_refer_s} \u03a9'
                out_x_l_eq_refer_s = f'{x_l_eq_refer_s}j \u03a9'

                window['-OUT_Z_EQ_S-'].update(out_z_eq_refer_s)
                window['-OUT_R_EQ_S-'].update(out_r_eq_refer_s)
                window['-OUT_X_L_EQ_S-'].update(out_x_l_eq_refer_s)


                # call parallel impedance function, return and display 6 outputs
                # parallel impedance retrieved is that referred to secondary
                # since HV primary is open circuited and values are referred to secondary
                # refer_to_p function is called to calculate values referred to primary
                g_phi, b_phi, y_phi_mag, y_phi_ang, r_phi, x_phi, z_phi_mag, z_phi_ang = calc_z_phi(
```

```
                    values['-INP_V_OC-'], values['-INP_I_OC-'], values['-INP_P_OC-']
                    )

                y_phi_polar = f'{y_phi_mag} \u2220 {y_phi_ang} \u03a9'
                y_phi_rect = f'{g_phi}{b_phi:+}j \u03a9'

                z_phi_polar = f'{z_phi_mag} \u2220 {z_phi_ang} \u03a9'
                z_phi_rect = f'{r_phi}{x_phi:+}j \u03a9'

                out_y_phi = f'{y_phi_polar}\n{y_phi_rect}'
                out_g_phi = f'{g_phi} \u03a9'
                out_b_phi = f'{b_phi}j \u03a9'
                out_z_phi = f'{z_phi_polar}\n{z_phi_rect}'
                out_r_phi = f'{r_phi} \u03a9'
                out_x_phi = f'{x_phi}j \u03a9'

                window['-OUT_Y_PHI_S-'].update(out_y_phi)
                window['-OUT_G_PHI_S-'].update(out_g_phi)
                window['-OUT_B_PHI_S-'].update(out_b_phi)
                window['-OUT_Z_PHI_S-'].update(out_z_phi)
                window['-OUT_R_PHI_S-'].update(out_r_phi)
                window['-OUT_X_PHI_S-'].update(out_x_phi)

                y_phi_mag_refer_p = refer_to_p(y_phi_mag, t_ratio)
                g_phi_refer_p = refer_to_p(g_phi, t_ratio)
                b_phi_refer_p = refer_to_p(b_phi, t_ratio)
                z_phi_mag_refer_p = refer_to_p(z_phi_mag, t_ratio)
                r_phi_refer_p = refer_to_p(r_phi, t_ratio)
                x_phi_refer_p = refer_to_p(x_phi, t_ratio)

                y_phi_polar_refer_p = f'{y_phi_mag_refer_p} \u2220 {y_phi_ang} \u03a9'
                y_phi_rect_refer_p = f'{g_phi_refer_p}{b_phi_refer_p:+}j \u03a9'

                z_phi_polar_refer_p = f'{z_phi_mag_refer_p} \u2220 {z_phi_ang} \u03a9'
                z_phi_rect_refer_p = f'{r_phi_refer_p}{x_phi_refer_p:+}j \u03a9'

                out_y_phi_refer_p = f'{y_phi_polar_refer_p}\n{y_phi_rect_refer_p}'
                out_g_phi_refer_p = f'{g_phi_refer_p} \u03a9'
                out_b_phi_refer_p = f'{b_phi_refer_p}j \u03a9'
                out_z_phi_refer_p = f'{z_phi_polar_refer_p}\n{z_phi_rect_refer_p}'
                out_r_phi_refer_p = f'{r_phi_refer_p} \u03a9'
                out_x_phi_refer_p = f'{x_phi_refer_p}j \u03a9'

                window['-OUT_Y_PHI_P-'].update(out_y_phi_refer_p)
                window['-OUT_G_PHI_P-'].update(out_g_phi_refer_p)
                window['-OUT_B_PHI_P-'].update(out_b_phi_refer_p)
                window['-OUT_Z_PHI_P-'].update(out_z_phi_refer_p)
```

```
                window['-OUT_R_PHI_P-'].update(out_r_phi_refer_p)
                window['-OUT_X_PHI_P-'].update(out_x_phi_refer_p)


                # call voltage regulation function, return outputs
                i_s, v_s_fl, v_s_nl, vr, vr_lag, vr_lead, i_s_range, vr_range, \
                    i_s_lag_range, vr_lag_range, i_s_lead_range, vr_lead_range = calc_vr(
                    values['-INP_S-'], values['-INP_V_P-'], values['-INP_V_S-'], r_eq, x_l_eq
                    )
                out_i_s = f'{i_s} A'
                out_v_s_fl = f'{v_s_fl} V'
                out_v_s_nl = f'{v_s_nl} V'
                out_vr = f'{vr} %'
                out_vr_lag = f'{vr_lag} %'
                out_vr_lead = f'{vr_lead} %'

                window['-OUT_I_S_RATED-'].update(out_i_s)
                window['-OUT_V_S_FL-'].update(out_v_s_fl)
                window['-OUT_V_S_NL-'].update(out_v_s_nl)
                window['-OUT_VR-'].update(out_vr)
                window['-OUT_VR_LAG-'].update(out_vr_lag)
                window['-OUT_VR_LEAD-'].update(out_vr_lead)

                # plot voltage regulation
                vr_plot(i_s_range, vr_range,
                            i_s_lag_range, vr_lag_range,
                            i_s_lead_range, vr_lead_range
                            )


                # call efficiency function
                pow_in, pow_out, loss_cu, loss_core, eff, eff_range, \
                    i_s_range_eff, loss_cu_range_perc, loss_core_range_perc = calc_eff(
                    values['-INP_S-'], values['-INP_V_S-'], i_s, v_s_nl, r_eq, r_phi
                )
                out_pow_in = f'{pow_in} W'
                out_pow_out = f'{pow_out} W'
                out_loss_cu = f'{loss_cu} W'
                out_loss_core = f'{loss_core} W'
                out_eff = f'{eff} %'

                window['-OUT_P_IN-'].update(out_pow_in)
                window['-OUT_P_OUT-'].update(out_pow_out)
                window['-OUT_P_CU-'].update(out_loss_cu)
                window['-OUT_P_C-'].update(out_loss_core)
                window['-OUT_EFF-'].update(out_eff)
```

```python

                # plot efficiency
                eff_plot(i_s_range_eff, eff_range, loss_cu_range_perc, loss_core_range_perc)



            else:
                prompt('Enter valid input!')


        else:
            prompt("Fill all the fields!")



window.close()

```