Electrical circuit solution program

Computer and control department

Second year

Presented to:

Dr/ Islam Shaalan

Dr / Osama Refaat

Presented by:

Amira Yehia

Doaa Helmy

Rovan Hussien

Huda Yasser

Abstract

Our program can solve an electrical circuit by take the needs from the user then draw the circuit and analysis it . In this program we use KCL and NODEL analysis to solve the circuit.

Disclaimer

We declare the following to be our own work, unless otherwise referenced, as defined by the University's policy on plagiarism.

Acknowledgments

Thanks are extended to the Faculty of Engineering Port Said University and Computer and Control Department for giving us a way to self-learn and improve ourselves. Thanks, are also extended to the Flexible Learning (Teaching and Learning) from our Advisors and Professors Dr/ Islam Shaalan and Dr / Osama Refaat. Special thanks goes to our parents who supported and encouraged us to complete this project on time. We also cannot forget to thank our friends, who have always been with us to complete this project.

Report content:

1 Introduction

Introduce a solution for electrical circuits in python language that solve , draw and analysis any complex circuit .
This program offers a comprehensive platform for analyzing circuits. It provides solutions, simulations, and visual representations, aiding both learners and professionals in understanding and troubleshooting diverse electric circuit configurations.

2 Packages

```
1  import SchemDraw as schem
2  import SchemDraw.elements as e
3  from tkinter import *
4  from tkinter.ttk import *
5  from tkinter import scrolledtext
6  import sympy
7  import numpy as np
8  from tkinter import messagebox
```

Fig1.imported libraries

We use these packages (fig1) to help us draw and solve the circuit

1. **SchemDraw** : is used for creating electronic circuit schematics. The "SchemDraw" library allows users to generate schematic diagrams in Python code. It provides a simple and intuitive way to create circuit drawings, making it useful for educational purposes, documentation, or presentations involving electronic circuit
2. **SchemDraw.elements:** SchemDraw provides a variety of predefined elements (components) that you can use in your circuit drawings. Elements include resistors, capacitors, inductors, voltage sources, grounds, etc.
3. **Tkinter:** introduces you to the exciting world of GUI(graph user interface) programming in Python.Tcl (pronounced as tickle) is a scripting language frequently used in testing, prototyping, and GUI development.on the other hand, is an open-source, cross-platform widget toolkit utilized by various programming languages to construct GUI programs. Tkinter allows you to develop desktop applications, making it a valuable tool for GUI programming in Python.It is a preferred choice for the following reasons: Easy to learn.Make a functional desktop application with minimal code. Layered design. Portable across all operating systems, including Windows, macOS, and Linux .
4. **SymPy, NumPy:** Utilized for symbolic mathematics, numerical operations, and scientific computing, respectively. Numerical and Symbolic Analysis: Utilize numerical techniques and symbolic computation libraries (like SymPy) for solving circuit equations and obtaining node voltages, branch currents, and other electrical parameters.

3 GUI (graphical user interface)

```
class MyWindow:
    def __init__(self, win):
        self.mylabel1 = Label(win,
                              text="Hello^_^!\n\n      welcome to our simple program for solving electric circuit "
                                   ",here a simple tips you should know")
        self.mylabel1.place(x=0, y=0)
        self.bb = Button(win, text="click here", command=self.our_tips)
        self.bb.place(x=420, y=60)
        self.lbl1 = Label(win, text='Enter the Start Point')
        self.lbl2 = Label(win, text='Enter the End Point')
        self.lbl4 = Label(win, text='Enter the Value')
        self.lbl5 = Label(win, text='Select the component')
        self.Start_Point = Entry()
        self.End_Point = Entry()
        self.Value = Entry()
        self.b1 = Button(win, text='Set Value', command=self.set)
        self.b2 = Button(win, text='Add the element', command=self.add)
        self.b3 = Button(win, text='Draw the circuit', command=self.drawCircuit)
        self.b3.place(x=400, y=450)
        self.b4 = Button(win, text='Analyze the circuit', command=self.analyze)
        self.b4.place(x=400, y=500)
        self.menubar = Menu(win)
        self.menubar.add_command(label="New", command=self.newCircuit)
        win.config(menu=self.menubar)
        self.b1.place(x=30, y=450)
        self.lbl5.place(x=20, y=100)
        data = ("Resistor", "Voltage Independent Source", "Current Independent Source", "Wire")
        self.cb = Combobox(win, values=data)
        self.cb.place(x=150, y=130, width=200)
        self.txt = scrolledtext.ScrolledText(win)
        self.elements = []
        self.results = []

1 usage
def our_tips(self):
    messagebox.showinfo( title: "tips",
                         message: "you should know some information about how to draw the circuit\n "
                        "->First:\nthe points are represented as 3x3 grid \n"
                        " ->Second:\ngrid[0][0] represent point 1\ngrid[0][1] represent point 4\n"
                        "grid[0][2] represent point 7\ngrid[1][0] represent point 2\n"
                        "grid[1][1] represent point 5\ngrid[1][2] represent point 8\ng"
                        "rid[2][0] represent point 3\ngrid[2][1] represent point 6\ngrid[2][2] represent point 9")
```

fig2Gui

It is a type of user interface that allows users to interact with electronic devices or software through graphical elements such as icons, buttons, and windows, rather than through text-based interfaces like command-line interfaces (CLI). GUIs(fig2) are designed to be more intuitive and user-friendly,

making it easier for users to perform tasks without needing to memorize complex commands.

Key components of a GUI include:

Windows: GUIs typically use windows to display information or applications. Each window may contain various graphical elements and controls.
Icons: Icons are small graphical representations of files, applications, or actions. Clicking on an icon often launches a corresponding application or opens a file.
Buttons: Buttons are clickable elements that perform a specific action when activated. They are often labeled with text or symbols to indicate their function.
Menus: Menus provide a list of options that users can select to perform various tasks. Menus can be displayed in a horizontal or vertical format and can contain sub-menus for more options.
Text Input: While GUIs are graphical in nature, they often include text input elements for tasks like typing text in a word processor or entering data in a form.
Graphics: GUIs use graphics to enhance the visual experience. This includes images, icons, and other visual elements.

So, in our program we use GUI as follows:

first in (fig3) a note or guide appear in a text box it helps you to tell me data about the grid and how the grid of drawing in our program used.

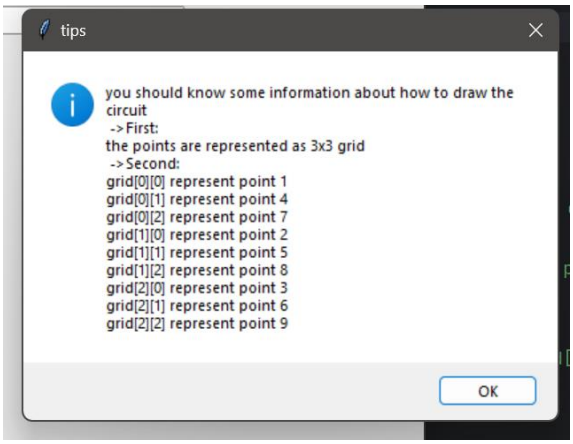


Fig3.tips

Second(fig4) combo box and we chose from it what we want.

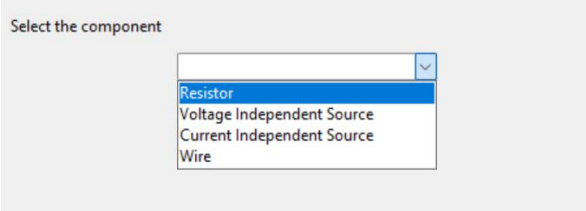


Fig4

Third there is 4 buttons(fig.5) that start and make the circuit:

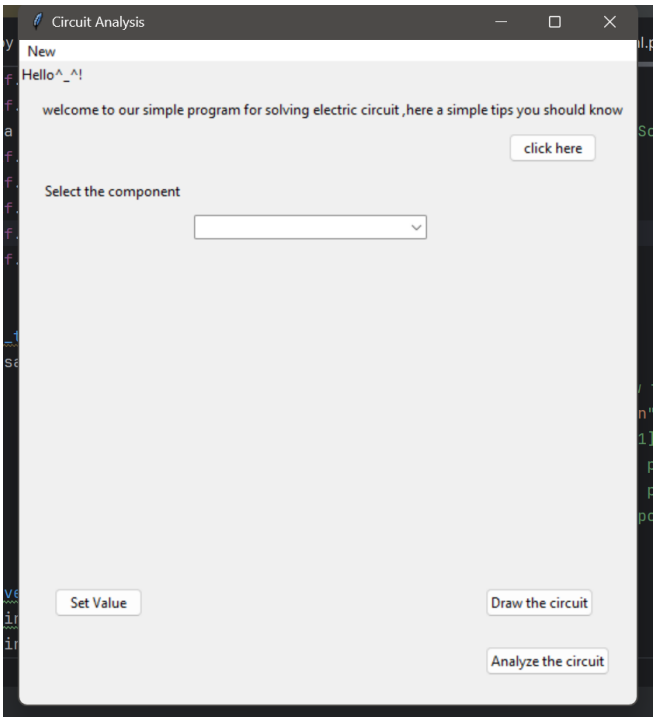


Fig.5

Set value: when we click on it, it show the start , end and value(fig6)

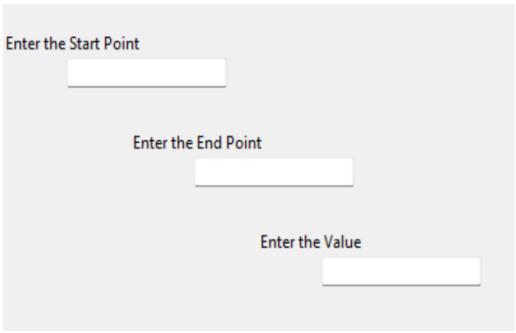


Fig.6

Add element: it can put a new element.

```
def add(self):
    self.b2.place(x=-100, y=-500)
    self.lbl1.place(x=-100, y=-350)
    self.lbl2.place(x=-100, y=-400)
    self.Start_Point.place(x=-200, y=-350)
    self.End_Point.place(x=-200, y=-400)
    self.Value.place(x=-200, y=-200)
    self.lbl4.place(x=-100, y=-200)

    sp = self.Start_Point.get()
    ep = self.End_Point.get()
    position = sp + ep
    kind = self.cb.get()
    if (kind == "Resistor" or kind == "Voltage Independent Source" or kind == "Current Independent Source"):
        value = int(self.Value.get())
        elem = element(kind, position, value)
    else:
        elem = element(kind, position)
    self.elements.append(elem)
```

Fig.7

4.Draw circuit:

The circuit drawn when we put these values(fig.8) it produce this (fig.9).

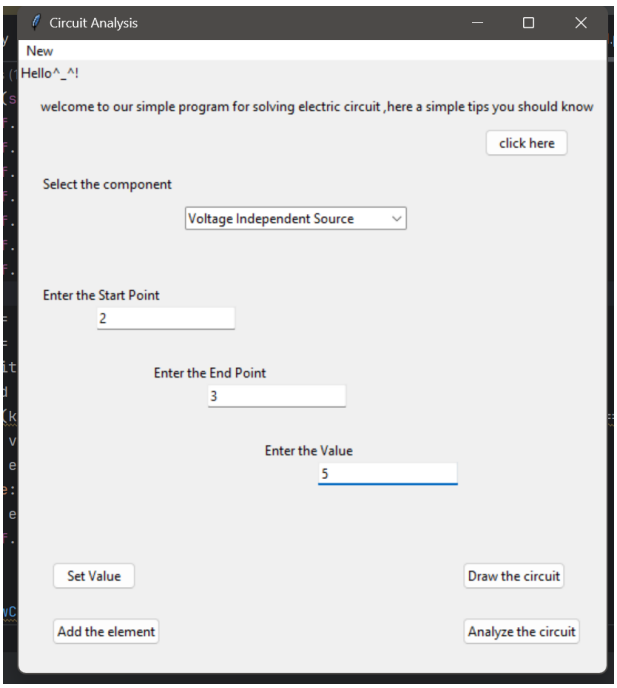


Fig.8

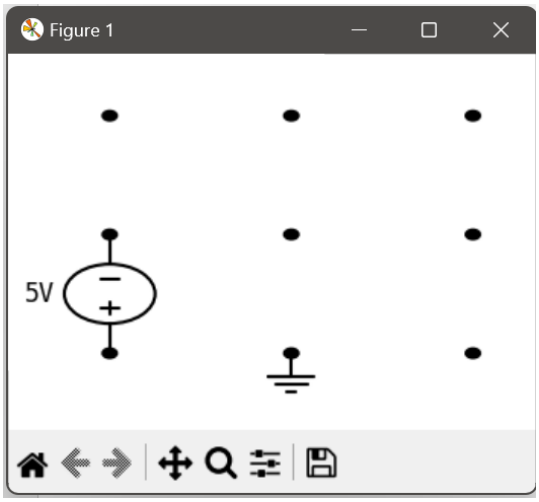


Fig.9

This our code for this part (fig.10)

```
def drawCircuit(self):
    d = schem.Drawing(unit=2, fontsize=12, font='monospace')
    d1 = d.add(e.DOT)
    l12 = d.add(e.LINE, d='down', xy=d1.end, color='white')
    d2 = d.add(e.DOT, xy=l12.end)
    l23 = d.add(e.LINE, d='down', xy=d2.end, color='white')
    d3 = d.add(e.DOT, xy=l23.end)
    l14 = d.add(e.LINE, d='right', xy=d1.end, color='white')
    d4 = d.add(e.DOT, xy=l14.end)
    l45 = d.add(e.LINE, d='down', xy=d4.end, color='white')
    d5 = d.add(e.DOT, xy=l45.end)
    l56 = d.add(e.LINE, d='down', xy=d5.end, color='white')
    d6 = d.add(e.DOT, xy=l56.end)
    l47 = d.add(e.LINE, d='right', xy=d4.end, color='white')
    d7 = d.add(e.DOT, xy=l47.end)
    l78 = d.add(e.LINE, d='down', xy=d7.end, color='white')
    d8 = d.add(e.DOT, xy=l78.end)
    l89 = d.add(e.LINE, d='down', xy=d8.end, color='white')
    d9 = d.add(e.DOT, xy=l89.end)
    l25 = d.add(e.LINE, d='right', xy=d2.end, color='white')
    l58 = d.add(e.LINE, d='right', xy=d5.end, color='white')
    l36 = d.add(e.LINE, d='right', xy=d3.end, color='white')
```

```
for elem in self.elements:
    if elem.position == '12':
        if (elem.kind == 'Resistor'):
            R1 = d.add(e.RES, d='down', label=str(elem.value) + '$\Omega$', xy=l12.start)
        elif (elem.kind == 'Voltage Independent Source'):
            S1 = d.add(e.SOURCE_V, d='down', label=str(elem.value) + 'V', xy=l12.start)
        elif (elem.kind == 'Current Independent Source'):
            S1 = d.add(e.SOURCE_I, d='down', label=str(elem.value) + 'A', xy=l12.start)
        l12.color = 'none'
        if (elem.kind == 'Wire'):
            l12.color = 'black'
    elif elem.position == '21':
        if (elem.kind == 'Resistor'):
            R1 = d.add(e.RES, d='up', label=str(elem.value) + '$\Omega$', xy=l12.end)
        elif (elem.kind == 'Voltage Independent Source'):
            S1 = d.add(e.SOURCE_V, d='up', label=str(elem.value) + 'V', xy=l12.end)
        elif (elem.kind == 'Current Independent Source'):
            S1 = d.add(e.SOURCE_I, d='up', label=str(elem.value) + 'A', xy=l12.end)
        l12.color = 'none'
        if (elem.kind == 'Wire'):
            l12.color = 'black'
```

Fig.10

5.Analyze the circuit

```
def _addKCLequations(self):
    for node in self.nodeList:
        if node != 0:
            equation = sympy.Rational( p: 0,  q: 1)
            for cm in self.components:
                if cm['k'] == 'r':
                    equation = self._addRtoNode(cm, equation, node)
                elif cm['k'] == 'vs' or cm['k'] == 'cvs':
                    equation = self._addVtoNode(cm, equation, node)
                elif cm['k'] == 'is' or cm['k'] == 'cis':
                    equation = self._addItoNode(cm, equation, node)
            self.equations.append(equation)
```

Fig.11

```
#solve the equation
    1 usage
    def _solveEquations(self):
        self.sSolution = sympy.solve(self.equations,  *symbols: list(self.unknowns))
#return the value not sump
    1 usage
    def _nameSolution(self):
        self.solution = {}
        for sym in self.sSolution:
            key = self.name[sym]
            self.solution[key] = self.sSolution[sym]

    1 usage
    def _substituteSolution(self):
        self.particular = {}
        for key in self.solution:
            self.particular[key] = self.solution[key].subs(self.subsDic)
```

Fig.12

```
1 usage
def _addVequations(self):

    for cm in self.components:
        if cm['k'] == 'vs' or cm['k'] == 'cvs':

            self.unknowns.add(cm['isy'])
            n1 = cm['n1']
            n2 = cm['n2']
            if n1 == 0:
                self.equations.append(sympy.Eq(cm['sy'], self.nodeVars[n2]))
            elif n2 == 0:
                self.equations.append(sympy.Eq(cm['sy'], self.nodeVars[n1]))
            else:
                self.equations.append(sympy.Eq(cm['sy'], self.nodeVars[n2] - self.nodeVars[n1]))
```

Fig.13

```
1 usage
def solve(self):

    self.equations = []
    self.unknowns = set([])
    self._numNodes()
    self._nodeVariables()
    self._addKCLequations()
    self._addVequations()
    self._solveEquations()
    self._nameSolution()
    self._substituteSolution()
    return self.particular
```

Fig.15

```
1 usage
def _addRtoNode(self, res, eq, node):
    n1 = res['n1']
    n2 = res['n2']
    if n1 == node:
        if n2 == 0:
            eq = eq - self.nodeVars[n1] / res['sy']
        else:
            eq = eq - (self.nodeVars[n1] - self.nodeVars[n2]) / res['sy']
    if n2 == node:
        if n1 == 0:
            eq = eq - self.nodeVars[n2] / res['sy']
        else:
            eq = eq - (self.nodeVars[n2] - self.nodeVars[n1]) / res['sy']
    return eq
```

Fig,14

Explanation of each function:

1**. __init__(self)**
This function initializes the circuit object. It sets up various attributes like components, subsDic, meas, etc., which are dictionaries/lists for storing components, substitution dictionaries, measurements, and solutions.

2**. addR(self, name, node1, node2, value=None)**
- *Purpose:* Adds a resistor to the circuit.
- *Parameters:*
  - name: Name/label of the resistor.
  - node1 and node2: Nodes to which the resistor is connected.
  - value: Optional resistance value.

3. **addV(self, name, node1, node2, value=None)**
- *Purpose:* Adds a voltage source to the circuit.
- *Parameters:*
  - name: Name/label of the voltage source.
  - node1 and node2: Nodes to which the voltage source is connected.
  - value: Optional voltage value.

4. addI(self, name, node1, node2, value=None)
- *Purpose:* Adds a current source to the circuit.
- *Parameters:*
  - name: Name/label of the current source.
  - node1 and node2: Nodes to which the current source is connected.
  - value: Optional current value.

5. _numNodes(self)
- *Purpose:* Determines the nodes present in the circuit.

6. _nodeVariables(self)
- *Purpose:* Assigns symbols to nodes, representing their voltages.

7. _addRtoNode(self, res, eq, node)
- *Purpose:* Adds resistors to the node equations based on their connections.

Similarly, functions _addVtoNode, _addItoNode, and _addIMtoNode handle equations related to voltage sources, current sources, and current-dependent sources (if defined).

8. _addKCLequations(self)
- *Purpose:* Creates equations based on Kirchhoff's Current Law (KCL) for each node.

9. _substEqs(self, oldS, newS)
- *Purpose:* Substitutes old symbols with new symbols in the equations.

10. _addVequations(self)

- *Purpose:* Adds equations specific to voltage sources to the circuit equations.

11. _showEquations(self)
- *Purpose:* Displays the circuit equations.

12. _solveEquations(self)
- *Purpose:* Solves the system of equations.

13. _nameSolution(self)
- *Purpose:* Assigns names to the solved symbols.

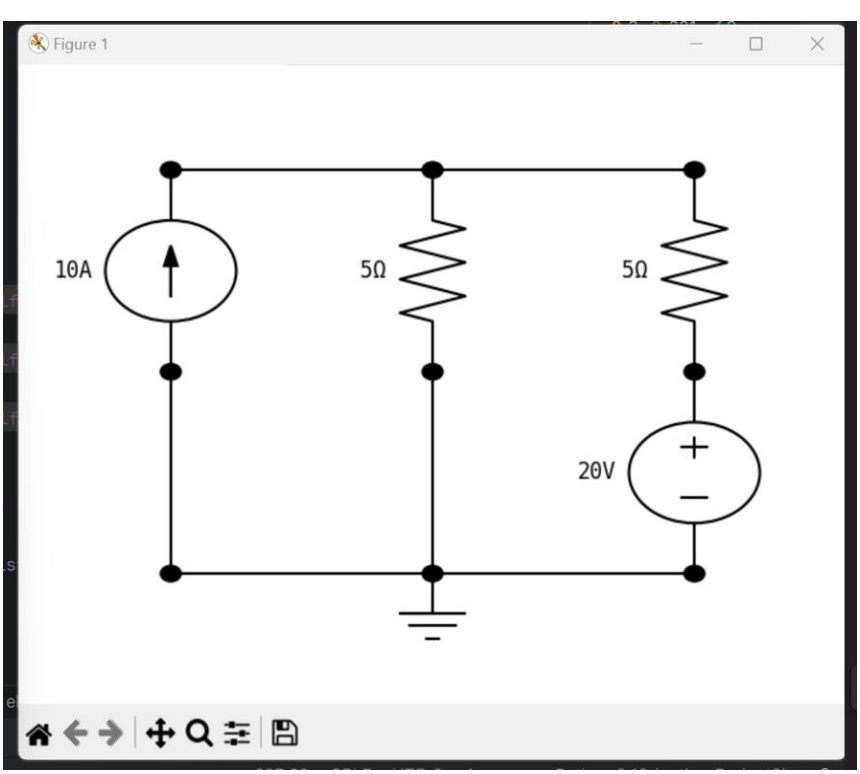
14. _substituteSolution(self)
- *Purpose:* Substitutes the solution values into the equations.

15. solve(self)
- *Purpose:* Orchestrates the solving process and returns the computed particular solution.

16. subs(self)
- *Purpose:* Returns the computed particular solution.

6.Some examples to a circuit and its analysis.

Example 1(fig16)



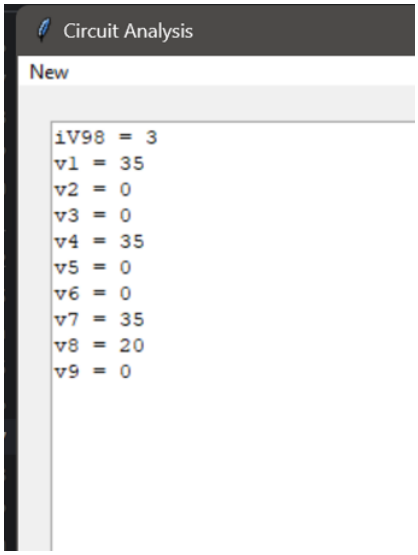
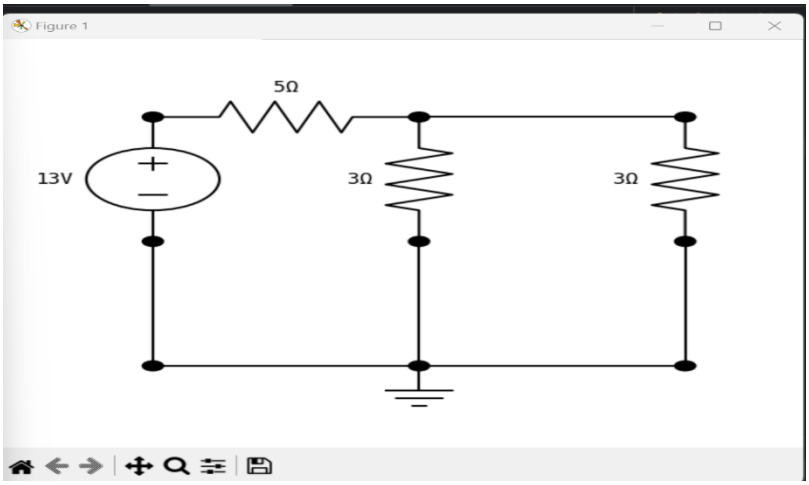


Fig.16

Example 2(fig17)



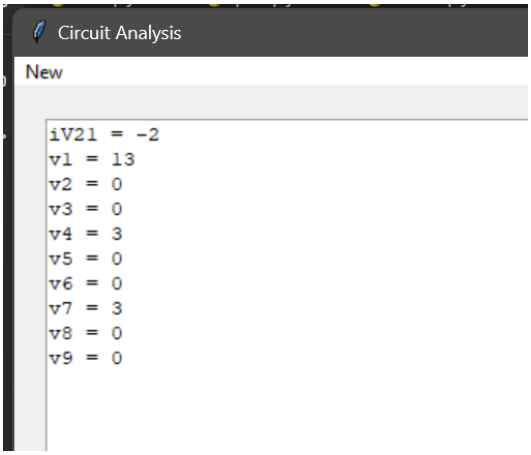
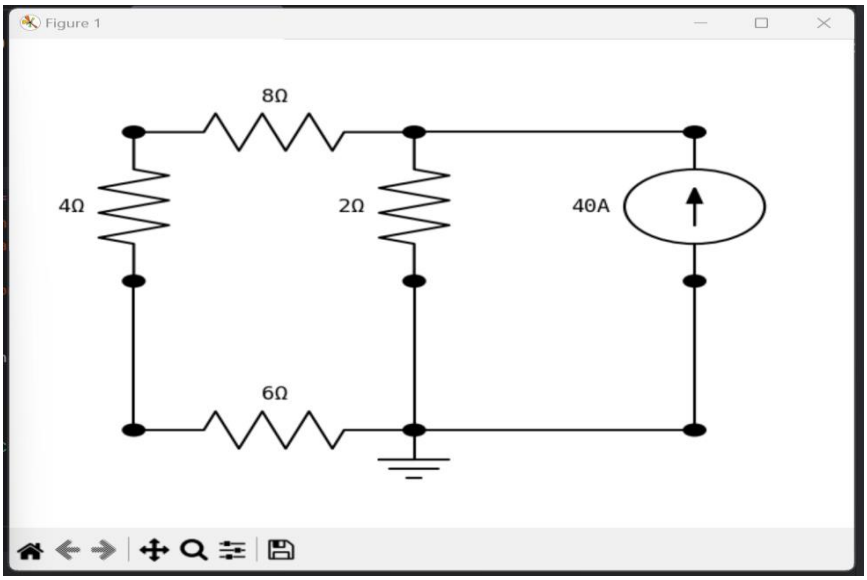


Fig.17

Example 3(fig.18)



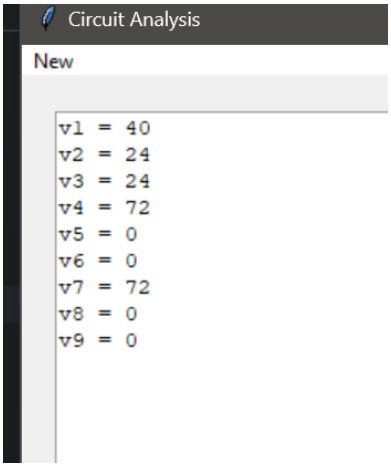


Fig.18

7.Final code :

```
from __future__ import print_function
from __future__ import division
import SchemDraw as schem
import SchemDraw.elements as e
from tkinter import *
from tkinter.ttk import *
from tkinter import scrolledtext
import sympy
import numpy as np
from tkinter import messagebox
from PIL import ImageTk, Image
from IPython.display import display, Math

class element:
    def __init__(self, kind, position, value=0):
        self.kind = kind
        self.position = position
        self.value = value

class MyWindow:
    def __init__(self, win):
        self.mylabel1 = Label(win,
                              text="Hello^_^!\n\n      welcome to our simple program for solving electric circuit
"
                                   ",here a simple tips you should know")
        self.mylabel1.place(x=0, y=0)
        self.bb = Button(win, text="click here", command=self.our_tips)
        self.bb.place(x=420, y=60)
        self.lbl1 = Label(win, text='Enter the Start Point')
        self.lbl2 = Label(win, text='Enter the End Point')
        self.lbl4 = Label(win, text='Enter the Value')
        self.lbl5 = Label(win, text='Select the component')
        self.Start_Point = Entry()
        self.End_Point = Entry()
        self.Value = Entry()
        self.b1 = Button(win, text='Set Value', command=self.set)
        self.b2 = Button(win, text='Add the element', command=self.add)
        self.b3 = Button(win, text='Draw the circuit', command=self.drawCircuit)
        self.b3.place(x=400, y=450)
        self.b4 = Button(win, text='Analyze the circuit', command=self.analyze)
        self.b4.place(x=400, y=500)
        self.menubar = Menu(win)
        self.menubar.add_command(label="New", command=self.newCircuit)
        win.config(menu=self.menubar)
        self.b1.place(x=30, y=450)
        self.lbl5.place(x=20, y=100)
        data = ("Resistor", "Voltage Independent Source", "Current Independent Source", "Wire")
```

```
        self.cb = Combobox(win, values=data)
        self.cb.place(x=150, y=130, width=200)
        self.txt = scrolledtext.ScrolledText(win)
        self.elements = []
        self.results = []

    def our_tips(self):
        messagebox.showinfo("tips",
                            "you should know some information about how to draw the circuit\n "
                            "->First:\nthe points are represented as 3x3 grid \n"
                            " ->Second:\ngrid[0][0] represent point 1\ngrid[0][1] represent point 4\n"
                            "grid[0][2] represent point 7\ngrid[1][0] represent point 2\n"
                            "grid[1][1] represent point 5\ngrid[1][2] represent point 8\ng"
                            "rid[2][0] represent point 3\ngrid[2][1] represent point 6\ngrid[2][2] represent point
9")

    def solvecircuit(self):
        mycircuit = circuit()
        mycircuit.addR('R0', 0, 6, 0)
        for elem in self.elements:
            if elem.kind == "Wire":
                mycircuit.addR('R' + elem.position, int(elem.position[0]), int(elem.position[1]), 0)

            elif elem.kind == "Resistor":
                mycircuit.addR('R' + elem.position, int(elem.position[0]), int(elem.position[1]),
elem.value)

            elif elem.kind == "Voltage Independent Source":
                mycircuit.addV('V' + elem.position, int(elem.position[0]), int(elem.position[1]),
elem.value)

            elif elem.kind == "Current Independent Source":
                mycircuit.addI('I' + elem.position, int(elem.position[0]), int(elem.position[1]), elem.value)

            self.results = mycircuit.solve()

    def analyze(self):
        self.mylabel1.place(x=-100, y=-100)
        self.bb.place(x=-100, y=-100)
        self.b2.place(x=-100, y=-500)
        self.lbl1.place(x=-100, y=-350)
        self.lbl2.place(x=-100, y=-400)
        self.Start_Point.place(x=-200, y=-350)
        self.End_Point.place(x=-200, y=-400)
        self.Value.place(x=-200, y=-200)
        self.lbl4.place(x=-100, y=-200)
        self.lbl5.place(x=-100, y=-200)
        self.cb.place(x=-100, y=-200)
        self.b1.place(x=-300, y=-500)
        self.b2.place(x=-300, y=-500)
        self.b3.place(x=-300, y=-500)
```

```
        self.b4.place(x=-300, y=-500)
        self.solvecircuit()
        for result in self.results:
            self.txt.insert(INSERT, result)
            self.txt.insert(INSERT, " = ")
            self.txt.insert(INSERT, self.results[result])
            self.txt.insert(INSERT, '\n')
        self.txt.place(x=20, y=20)

    def newCircuit(self):
        self.bb.place(x=420, y=60)
        self.mylabel1.place(x=0, y=0)
        self.cb.place(x=150, y=130, width=200)
        self.lbl5.place(x=20, y=100)
        self.b3.place(x=400, y=450)
        self.b4.place(x=400, y=500)
        self.b1.place(x=30, y=450)
        self.txt.place(x=-1000, y=-1000)
        for elem in self.elements:
            self.elements.remove(elem)
        self.b2.place(x=-100, y=-500)
        self.lbl1.place(x=-100, y=-350)
        self.lbl2.place(x=-100, y=-400)
        self.Start_Point.place(x=-200, y=-350)
        self.End_Point.place(x=-200, y=-400)
        self.Value.place(x=-200, y=-200)
        self.lbl4.place(x=-100, y=-200)

    def set(self):
        self.cb.place(x=150, y=130, width=200)
        self.lbl5.place(x=20, y=100)
        self.b3.place(x=400, y=450)
        self.b4.place(x=400, y=500)
        self.b1.place(x=30, y=450)
        self.txt.place(x=-1000, y=-1000)
        self.b2.place(x=-100, y=-500)
        self.lbl1.place(x=-100, y=-350)
        self.lbl2.place(x=-100, y=-400)
        self.Start_Point.place(x=-200, y=-350)
        self.End_Point.place(x=-200, y=-400)
        self.Value.place(x=-200, y=-200)
        self.lbl4.place(x=-100, y=-200)

        self.Start_Point.delete(0, 'end')
        self.End_Point.delete(0, 'end')
        self.Value.delete(0, 'end')

        kind = self.cb.get()
        self.lbl1.place(x=20, y=200)
        self.Start_Point.place(x=70, y=220)
        self.lbl2.place(x=120, y=270)
```

```
        self.End_Point.place(x=170, y=290)
        self.b2.place(x=30, y=500)

        if (kind == "Resistor" or kind == "Voltage Independent Source" or kind == "Current
Independent Source"):
            self.Value.place(x=270, y=360)
            self.lbl4.place(x=220, y=340)

    def add(self):
        self.b2.place(x=-100, y=-500)
        self.lbl1.place(x=-100, y=-350)
        self.lbl2.place(x=-100, y=-400)
        self.Start_Point.place(x=-200, y=-350)
        self.End_Point.place(x=-200, y=-400)
        self.Value.place(x=-200, y=-200)
        self.lbl4.place(x=-100, y=-200)

        sp = self.Start_Point.get()
        ep = self.End_Point.get()
        position = sp + ep
        kind = self.cb.get()
        if (kind == "Resistor" or kind == "Voltage Independent Source" or kind == "Current
Independent Source"):
            value = int(self.Value.get())
            elem = element(kind, position, value)
        else:
            elem = element(kind, position)
        self.elements.append(elem)

    def drawCircuit(self):
        d = schem.Drawing(unit=2, fontsize=12, font='monospace')
        d1 = d.add(e.DOT)
        l12 = d.add(e.LINE, d='down', xy=d1.end, color='white')
        d2 = d.add(e.DOT, xy=l12.end)
        l23 = d.add(e.LINE, d='down', xy=d2.end, color='white')
        d3 = d.add(e.DOT, xy=l23.end)
        l14 = d.add(e.LINE, d='right', xy=d1.end, color='white')
        d4 = d.add(e.DOT, xy=l14.end)
        l45 = d.add(e.LINE, d='down', xy=d4.end, color='white')
        d5 = d.add(e.DOT, xy=l45.end)
        l56 = d.add(e.LINE, d='down', xy=d5.end, color='white')
        d6 = d.add(e.DOT, xy=l56.end)
        l47 = d.add(e.LINE, d='right', xy=d4.end, color='white')
        d7 = d.add(e.DOT, xy=l47.end)
        l78 = d.add(e.LINE, d='down', xy=d7.end, color='white')
        d8 = d.add(e.DOT, xy=l78.end)
        l89 = d.add(e.LINE, d='down', xy=d8.end, color='white')
        d9 = d.add(e.DOT, xy=l89.end)
        l25 = d.add(e.LINE, d='right', xy=d2.end, color='white')
        l58 = d.add(e.LINE, d='right', xy=d5.end, color='white')
        l36 = d.add(e.LINE, d='right', xy=d3.end, color='white')
```

```
l69 = d.add(e.LINE, d='right', xy=d6.end, color='white')
gnd = d.add(e.GND, d='right', xy=l56.end, color='black')

for elem in self.elements:
    if elem.position == '12':
        if (elem.kind == 'Resistor'):
            R1 = d.add(e.RES, d='down', label=str(elem.value) + '$\Omega$', xy=l12.start)
        elif (elem.kind == 'Voltage Independent Source'):
            S1 = d.add(e.SOURCE_V, d='down', label=str(elem.value) + 'V', xy=l12.start)
        elif (elem.kind == 'Current Independent Source'):
            S1 = d.add(e.SOURCE_I, d='down', label=str(elem.value) + 'A', xy=l12.start)
        l12.color = 'none'
        if (elem.kind == 'Wire'):
            l12.color = 'black'
    elif elem.position == '21':
        if (elem.kind == 'Resistor'):
            R1 = d.add(e.RES, d='up', label=str(elem.value) + '$\Omega$', xy=l12.end)
        elif (elem.kind == 'Voltage Independent Source'):
            S1 = d.add(e.SOURCE_V, d='up', label=str(elem.value) + 'V', xy=l12.end)
        elif (elem.kind == 'Current Independent Source'):
            S1 = d.add(e.SOURCE_I, d='up', label=str(elem.value) + 'A', xy=l12.end)
        l12.color = 'none'
        if (elem.kind == 'Wire'):
            l12.color = 'black'
    elif elem.position == '23':
        if (elem.kind == 'Resistor'):
            R2 = d.add(e.RES, d='down', label=str(elem.value) + '$\0mega$', xy=l23.start)
        elif (elem.kind == 'Voltage Independent Source'):
            S2 = d.add(e.SOURCE_V, d='down', label=str(elem.value) + 'V', xy=l23.start)
        elif (elem.kind == 'Current Independent Source'):
            S2 = d.add(e.SOURCE_I, d='down', label=str(elem.value) + 'A', xy=l23.start)
        l23.color = 'none'
        if (elem.kind == 'Wire'):
            l23.color = 'black'
    elif elem.position == '32':
        if (elem.kind == 'Resistor'):
            R2 = d.add(e.RES, d='up', label=str(elem.value) + '$\0mega$', xy=l23.end)
        elif (elem.kind == 'Voltage Independent Source'):
            S2 = d.add(e.SOURCE_V, d='up', label=str(elem.value) + 'V', xy=l23.end)
        elif (elem.kind == 'Current Independent Source'):
            S2 = d.add(e.SOURCE_I, d='up', label=str(elem.value) + 'A', xy=l23.end)
        l23.color = 'none'
        if (elem.kind == 'Wire'):
            l23.color = 'black'
    elif elem.position == '58':
        if (elem.kind == 'Resistor'):
            R3 = d.add(e.RES, d='right', label=str(elem.value) + '$\0mega$', xy=l58.start)
        elif (elem.kind == 'Voltage Independent Source'):
            S3 = d.add(e.SOURCE_V, d='right', label=str(elem.value) + 'V', xy=l58.start)
        elif (elem.kind == 'Current Independent Source'):
            S3 = d.add(e.SOURCE_I, d='right', label=str(elem.value) + 'A', xy=l58.start)
```

```
        l58.color = 'none'
        if (elem.kind == 'Wire'):
            l58.color = 'black'
    elif elem.position == '85':
        if (elem.kind == 'Resistor'):
            R3 = d.add(e.RES, d='left', label=str(elem.value) + '$\Omega$', xy=l58.end)
        elif (elem.kind == 'Voltage Independent Source'):
            S3 = d.add(e.SOURCE_V, d='left', label=str(elem.value) + 'V', xy=l58.end)
        elif (elem.kind == 'Current Independent Source'):
            S3 = d.add(e.SOURCE_I, d='left', label=str(elem.value) + 'A', xy=l58.end)
        l58.color = 'none'
        if (elem.kind == 'Wire'):
            l58.color = 'black'
    elif elem.position == '14':
        if (elem.kind == 'Resistor'):
            R4 = d.add(e.RES, d='right', label=str(elem.value) + '$\Omega$', xy=l14.start)
        elif (elem.kind == 'Voltage Independent Source'):
            S4 = d.add(e.SOURCE_V, d='right', label=str(elem.value) + 'V', xy=l14.start)
        elif (elem.kind == 'Current Independent Source'):
            S4 = d.add(e.SOURCE_I, d='right', label=str(elem.value) + 'A', xy=l14.start)
        l14.color = 'none'
        if (elem.kind == 'Wire'):
            l14.color = 'black'
    elif elem.position == '41':
        if (elem.kind == 'Resistor'):
            R4 = d.add(e.RES, d='left', label=str(elem.value) + '$\Omega$', xy=l14.end)
        elif (elem.kind == 'Voltage Independent Source'):
            S4 = d.add(e.SOURCE_V, d='left', label=str(elem.value) + 'V', xy=l14.end)
        elif (elem.kind == 'Current Independent Source'):
            S4 = d.add(e.SOURCE_I, d='left', label=str(elem.value) + 'A', xy=l14.end)
        l14.color = 'none'
        if (elem.kind == 'Wire'):
            l14.color = 'black'
    elif elem.position == '47':
        if (elem.kind == 'Resistor'):
            R5 = d.add(e.RES, d='right', label=str(elem.value) + '$\Omega$', xy=l47.start)
        elif (elem.kind == 'Voltage Independent Source'):
            S5 = d.add(e.SOURCE_V, d='right', label=str(elem.value) + 'V', xy=l47.start)
        elif (elem.kind == 'Current Independent Source'):
            S5 = d.add(e.SOURCE_I, d='right', label=str(elem.value) + 'A', xy=l47.start)
        l47.color = 'none'
        if (elem.kind == 'Wire'):
            l47.color = 'black'
    elif elem.position == '74':
        if (elem.kind == 'Resistor'):
            R5 = d.add(e.RES, d='left', label=str(elem.value) + '$\Omega$', xy=l47.end)
        elif (elem.kind == 'Voltage Independent Source'):
            S5 = d.add(e.SOURCE_V, d='left', label=str(elem.value) + 'V', xy=l47.end)
        elif (elem.kind == 'Current Independent Source'):
            S5 = d.add(e.SOURCE_I, d='left', label=str(elem.value) + 'A', xy=l47.end)
        l47.color = 'none'
```

```
        if (elem.kind == 'Wire'):
            l47.color = 'black'
    elif elem.position == '45':
        if (elem.kind == 'Resistor'):
            R6 = d.add(e.RES, d='down', label=str(elem.value) + '$\Omega$', xy=l45.start)
        elif (elem.kind == 'Voltage Independent Source'):
            S6 = d.add(e.SOURCE_V, d='down', label=str(elem.value) + 'V', xy=l45.start)
        elif (elem.kind == 'Current Independent Source'):
            S6 = d.add(e.SOURCE_I, d='down', label=str(elem.value) + 'A', xy=l45.start)
        l45.color = 'none'
        if (elem.kind == 'Wire'):
            l45.color = 'black'
    elif elem.position == '54':
        if (elem.kind == 'Resistor'):
            R6 = d.add(e.RES, d='up', label=str(elem.value) + '$\Omega$', xy=l45.end)
        elif (elem.kind == 'Voltage Independent Source'):
            S6 = d.add(e.SOURCE_V, d='up', label=str(elem.value) + 'V', xy=l45.end)
        elif (elem.kind == 'Current Independent Source'):
            S6 = d.add(e.SOURCE_I, d='up', label=str(elem.value) + 'A', xy=l45.end)
        l45.color = 'none'
        if (elem.kind == 'Wire'):
            l45.color = 'black'
    elif elem.position == '56':
        if (elem.kind == 'Resistor'):
            R7 = d.add(e.RES, d='down', label=str(elem.value) + '$\Omega$', xy=l56.start)
        elif (elem.kind == 'Voltage Independent Source'):
            S7 = d.add(e.SOURCE_V, d='down', label=str(elem.value) + 'V', xy=l56.start)
        elif (elem.kind == 'Current Independent Source'):
            S7 = d.add(e.SOURCE_I, d='down', label=str(elem.value) + 'A', xy=l56.start)
        l56.color = 'none'
        if (elem.kind == 'Wire'):
            l56.color = 'black'
    elif elem.position == '65':
        if (elem.kind == 'Resistor'):
            R7 = d.add(e.RES, d='up', label=str(elem.value) + '$\Omega$', xy=l56.end)
        elif (elem.kind == 'Voltage Independent Source'):
            S7 = d.add(e.SOURCE_V, d='up', label=str(elem.value) + 'V', xy=l56.end)
        elif (elem.kind == 'Current Independent Source'):
            S7 = d.add(e.SOURCE_I, d='up', label=str(elem.value) + 'A', xy=l56.end)
        l56.color = 'none'
        if (elem.kind == 'Wire'):
            l56.color = 'black'
    elif elem.position == '78':
        if (elem.kind == 'Resistor'):
            R8 = d.add(e.RES, d='down', label=str(elem.value) + '$\Omega$', xy=l78.start)
        elif (elem.kind == 'Voltage Independent Source'):
            S8 = d.add(e.SOURCE_V, d='down', label=str(elem.value) + 'V', xy=l78.start)
        elif (elem.kind == 'Current Independent Source'):
            S8 = d.add(e.SOURCE_I, d='down', label=str(elem.value) + 'A', xy=l78.start)
        l78.color = 'none'
        if (elem.kind == 'Wire'):
```

```
            l78.color = 'black'
        elif elem.position == '87':
            if (elem.kind == 'Resistor'):
                R8 = d.add(e.RES, d='up', label=str(elem.value) + '$\Omega$', xy=l78.end)
            elif (elem.kind == 'Voltage Independent Source'):
                S8 = d.add(e.SOURCE_V, d='up', label=str(elem.value) + 'V', xy=l78.end)
            elif (elem.kind == 'Current Independent Source'):
                S8 = d.add(e.SOURCE_I, d='up', label=str(elem.value) + 'A', xy=l78.end)
            l78.color = 'none'
            if (elem.kind == 'Wire'):
                l78.color = 'black'
        elif elem.position == '36':
            if (elem.kind == 'Resistor'):
                R9 = d.add(e.RES, d='right', label=str(elem.value) + '$\Omega$', xy=l36.start)
            elif (elem.kind == 'Voltage Independent Source'):
                S9 = d.add(e.SOURCE_V, d='right', label=str(elem.value) + 'V', xy=l36.start)
            elif (elem.kind == 'Current Independent Source'):
                S9 = d.add(e.SOURCE_I, d='right', label=str(elem.value) + 'A', xy=l36.start)
            l36.color = 'none'
            if (elem.kind == 'Wire'):
                l36.color = 'black'
        elif elem.position == '63':
            if (elem.kind == 'Resistor'):
                R9 = d.add(e.RES, d='left', label=str(elem.value) + '$\Omega$', xy=l36.end)
            elif (elem.kind == 'Voltage Independent Source'):
                S9 = d.add(e.SOURCE_V, d='left', label=str(elem.value) + 'V', xy=l36.end)
            elif (elem.kind == 'Current Independent Source'):
                S9 = d.add(e.SOURCE_I, d='left', label=str(elem.value) + 'A', xy=l36.end)
            l36.color = 'none'
            if (elem.kind == 'Wire'):
                l36.color = 'black'
        elif elem.position == '69':
            if (elem.kind == 'Resistor'):
                R10 = d.add(e.RES, d='right', label=str(elem.value) + '$\Omega$', xy=l69.start)
            elif (elem.kind == 'Voltage Independent Source'):
                S10 = d.add(e.SOURCE_V, d='right', label=str(elem.value) + 'V', xy=l69.start)
            elif (elem.kind == 'Current Independent Source'):
                S10 = d.add(e.SOURCE_I, d='right', label=str(elem.value) + 'A', xy=l69.start)
            l69.color = 'none'
            if (elem.kind == 'Wire'):
                l69.color = 'black'
        elif elem.position == '96':
            if (elem.kind == 'Resistor'):
                R10 = d.add(e.RES, d='left', label=str(elem.value) + '$\Omega$', xy=l69.end)
            elif (elem.kind == 'Voltage Independent Source'):
                S10 = d.add(e.SOURCE_V, d='left', label=str(elem.value) + 'V', xy=l69.end)
            elif (elem.kind == 'Current Independent Source'):
                S10 = d.add(e.SOURCE_I, d='left', label=str(elem.value) + 'A', xy=l69.end)
            l69.color = 'none'
            if (elem.kind == 'Wire'):
                l69.color = 'black'
```

```
        elif elem.position == '89':
            if (elem.kind == 'Resistor'):
                R11 = d.add(e.RES, d='down', label=str(elem.value) + '$\Omega$', xy=l89.start)
            elif (elem.kind == 'Voltage Independent Source'):
                S11 = d.add(e.SOURCE_V, d='down', label=str(elem.value) + 'V', xy=l89.start)
            elif (elem.kind == 'Current Independent Source'):
                S11 = d.add(e.SOURCE_I, d='down', label=str(elem.value) + 'A', xy=l89.start)
            l89.color = 'none'
            if (elem.kind == 'Wire'):
                l89.color = 'black'
        elif elem.position == '98':
            if (elem.kind == 'Resistor'):
                R11 = d.add(e.RES, d='up', label=str(elem.value) + '$\Omega$', xy=l89.end)
            elif (elem.kind == 'Voltage Independent Source'):
                S11 = d.add(e.SOURCE_V, d='up', label=str(elem.value) + 'V', xy=l89.end)
            elif (elem.kind == 'Current Independent Source'):
                S11 = d.add(e.SOURCE_I, d='up', label=str(elem.value) + 'A', xy=l89.end)
            l89.color = 'none'
            if (elem.kind == 'Wire'):
                l89.color = 'black'
        elif elem.position == '25':
            if (elem.kind == 'Resistor'):
                R12 = d.add(e.RES, d='right', label=str(elem.value) + '$\Omega$', xy=l25.start)
            elif (elem.kind == 'Voltage Independent Source'):
                S12 = d.add(e.SOURCE_V, d='right', label=str(elem.value) + 'V', xy=l25.start)
            elif (elem.kind == 'Current Independent Source'):
                S12 = d.add(e.SOURCE_I, d='right', label=str(elem.value) + 'A', xy=l25.start)
            l25.color = 'none'
            if (elem.kind == 'Wire'):
                l25.color = 'black'
        elif elem.position == '52':
            if (elem.kind == 'Resistor'):
                R12 = d.add(e.RES, d='left', label=str(elem.value) + '$\Omega$', xy=l25.end)
            elif (elem.kind == 'Voltage Independent Source'):
                S12 = d.add(e.SOURCE_V, d='left', label=str(elem.value) + 'V', xy=l25.end)
            elif (elem.kind == 'Current Independent Source'):
                S12 = d.add(e.SOURCE_I, d='left', label=str(elem.value) + 'A', xy=l25.end)
            l25.color = 'none'
            if (elem.kind == 'Wire'):
                l25.color = 'black'
    d.draw()


s = sympy.Symbol('s')


class circuit():
    def __init__(self):
        self.components = []
        self.subsDic = {}
```

```
        self.meas = {}
        self.sSolution = None
        self.solution = None
        self.particular = None
        self.name = {}
        self.symbol = {}

    def addR(self, name, node1, node2, value=None):
        sy = sympy.Symbol(name)

        dict = {}
        dict['k'] = 'r'
        dict['n'] = name
        dict['n1'] = node1
        dict['n2'] = node2
        dict['v'] = value
        dict['sy'] = sy

        self.components.append(dict)

        self.symbol[name] = sy

        if value != None:
            self.subsDic[sy] = value
        return sy

    def addV(self, name, node1, node2, value=None):
        sy = sympy.Symbol(name)

        dict = {}
        dict['k'] = 'vs'
        dict['n'] = name
        dict['n1'] = node1
        dict['n2'] = node2
        dict['v'] = value
        dict['sy'] = sy

        isy = sympy.Symbol('i' + name)
        dict['isy'] = isy

        self.name[isy] = 'i' + name

        self.symbol[name] = sy
        self.symbol['i' + name] = isy

        self.components.append(dict)

        if value != None:
            self.subsDic[sy] = value
        return sy
```

```
def addI(self, name, node1, node2, value=None):
    sy = sympy.Symbol(name)

    dict = {}
    dict['k'] = 'is'
    dict['n'] = name
    dict['n1'] = node1
    dict['n2'] = node2
    dict['v'] = value
    dict['sy'] = sy
    self.components.append(dict)
    self.symbol[name] = sy

    if value != None:
        self.subsDic[sy] = value
    return sy

def _numNodes(self):
    self.nodeList = set([])

    for component in self.components:
        self.nodeList.add(component['n1'])
        self.nodeList.add(component['n2'])

    self.nodeList = list(self.nodeList)

def _nodeVariables(self):
    self.nodeVars = {}

    zeroFound = False
    for node in self.nodeList:
        if node == 0:
            zeroFound = True
        else:
            name = 'v' + str(node)
            ns = sympy.Symbol(name)
            self.nodeVars[node] = ns
            self.unknowns.add(ns)

            self.name[ns] = name

            self.symbol[name] = ns


def _addRtoNode(self, res, eq, node):
    n1 = res['n1']
    n2 = res['n2']
    if n1 == node:
        if n2 == 0:
            eq = eq - self.nodeVars[n1] / res['sy']
        else:
```

```
            eq = eq - (self.nodeVars[n1] - self.nodeVars[n2]) / res['sy']
        if n2 == node:
            if n1 == 0:
                eq = eq - self.nodeVars[n2] / res['sy']
            else:
                eq = eq - (self.nodeVars[n2] - self.nodeVars[n1]) / res['sy']
        return eq

    def _addVtoNode(self, vs, eq, node):
        n1 = vs['n1']
        n2 = vs['n2']
        if n1 == node:
            eq = eq + vs['isy']
        if n2 == node:
            eq = eq - vs['isy']
        return eq

    def _addItoNode(self, isr, eq, node):
        n1 = isr['n1']
        n2 = isr['n2']
        if n1 == node:
            eq = eq + isr['sy']
        if n2 == node:
            eq = eq - isr['sy']
        return eq
    def _addKCLequations(self):
        for node in self.nodeList:
            if node != 0:
                equation = sympy.Rational(0, 1)
                for cm in self.components:
                    if cm['k'] == 'r':
                        equation = self._addRtoNode(cm, equation, node)
                    elif cm['k'] == 'vs' or cm['k'] == 'cvs':
                        equation = self._addVtoNode(cm, equation, node)
                    elif cm['k'] == 'is' or cm['k'] == 'cis':
                        equation = self._addItoNode(cm, equation, node)
                self.equations.append(equation)

    def _substEqs(self, oldS, newS):
        newList = []
        for eq in self.equations:
            newList.append(eq.subs(oldS, newS))
        self.equations = newList

    def _addVequations(self):

        for cm in self.components:
            if cm['k'] == 'vs' or cm['k'] == 'cvs':

                self.unknowns.add(cm['isy'])
                n1 = cm['n1']
```

```
            n2 = cm['n2']
            if n1 == 0:
                self.equations.append(sympy.Eq(cm['sy'], self.nodeVars[n2]))
            elif n2 == 0:
                self.equations.append(sympy.Eq(cm['sy'], self.nodeVars[n1]))
            else:
                self.equations.append(sympy.Eq(cm['sy'], self.nodeVars[n2] - self.nodeVars[n1]))

    def _solveEquations(self):
        self.sSolution = sympy.solve(self.equations, list(self.unknowns))
#return the value not sump
    def _nameSolution(self):
        self.solution = {}
        for sym in self.sSolution:
            key = self.name[sym]
            self.solution[key] = self.sSolution[sym]

    def _substituteSolution(self):
        self.particular = {}
        for key in self.solution:
            self.particular[key] = self.solution[key].subs(self.subsDic)

    def solve(self):
        self.equations = []
        self.unknowns = set([])
        self._numNodes()
        self._nodeVariables()
        self._addKCLequations()
        self._addVequations()
        self._solveEquations()
        self._nameSolution()
        self._substituteSolution()
        return self.particular

    def subs(self):
        return self.particular

if __name__ == "__main__":
    window = Tk()
    mywin = MyWindow(window)
    window.title('Circuit Analysis')
    window.geometry("530x550")
    window.mainloop()
    elements = mywin.elements
```

8.References:

- **https://blog.hubspot.com/website/what-is-gui#:~:text=A%20graphical%20user%20interface%20(GUI,actions%20that%20they%20can%20take (a) (last visit at 2.1.2024).**
- **https://www.w3schools.com/python/ (b) (last visit at 23.12.2023).**
- **https://chat.openai.com/c/f345e8e7-02de-4e18-9b9e-c994806cec3f (c) (last visit at 3.1.2024).**
- **https://github.com/topics/electric-circuits (d) (last visit at 29.12.2023).**
- **https://www.youtube.com/playlist?list=PLDoPjvoNmBAyE_gei5d18qkfIe-Z8mocs (e) (last visit at 27.12.20)**