```
<!DOCTYPE html>
<html lang="en">
<head> <!--rechte Seite Formatierung-->
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Buck Converter Simulation</title>
  <style>
    canvas {
      border: 1px solid black;
    }
    .container {
      display: flex;
      flex-direction: column;
      align-items: left;
    }
    .input-container {
      margin-bottom: 10px;
    }

    .input-container label {
      display: block;
      margin-bottom: 5px;
    }

    .input-container input {
      width: 200px; /* Set a fixed width for the input fields */
    }
    .modeButton {
      display: block;
      width: 200px;
      padding: 15px;
      font-size: 16px;
```

```
      text-align: center;
      margin: 20px auto;
      border: none;
      border-radius: 5px;
      background-color: rgb(0, 89, 255);
      color: white;
      cursor: pointer;
    }
    #breakButton {
      background-color: red;
    }
    #cirquit_canvas, #plot {
      margin-bottom: 10px; /* Adjust as needed */
    }
    #modeButton.manual {
      background-color: grey;
    }
    #toggleButton.manual {
      background-color: grey;
    }
    #breakButton.run{
      background-color: green;
    }

  </style>
</head>

<!--html-Teil-->
<body>
  <div style="display: flex;">
    <div class="container">
      <canvas id="cirquit_canvas" width="600" height="400"></canvas>
```

```
    <canvas id="plot" width="600" height="300"></canvas>
  </div>
  <div style="margin-left: 20px;">
    <div class="input-container">
      <button id="modeButton" class="modeButton">Switch to Manual</button>
    </div>
    <div class="input-container">
      <button id="toggleButton" class="modeButton">SwitchOn</button>
    </div>
    <div class="input-container">
      <button id="restartButton" class="modeButton">Restart</button>
    </div>
    <div class="input-container">
      <button id="breakButton" class="modeButton">Break</button>
    </div>
    <div class="input-container">
      <label for="Resistor">Resistor (&Omega;):</label>
      <input type="number" id="resistanceInput" value="10"> <!--value hardgecodet, sollte möglicht der Wert von R sein-->
    </div>
    <div class="input-container">
      <label for="InputVoltage">Input Voltage (V):</label>
      <input type="number" id="voltageInput" value="20"> <!--value hardgecodet, sollte möglicht der Wert von V_in sein-->
    </div>
      <div class="input-container">
      <label for="Inductance">Inductance (mH):</label>
      <input type="number" id="inductanceInput" value="99" > <!--value hardgecodet, sollte möglicht der Wert von L sein-->
    </div>
    <div class="input-container">
      <label for="Capacitance">Capacitance (mF):</label>
```

```
        <input type="number" id="capacitanceInput" value="1"> <!--value hardgecodet, sollte möglicht der Wert von C sein-->
      </div>
      <div class="input-container">
        <label for="PeriodDuration"> Period Duration (ms):</label>
        <input type="number" id="periodDurationInput" value ="3"><!--value hardgecodet, sollte möglicht der Wert von T sein-->
      </div>
      <div class="input-container">
        <label for="DutyCycle">Duty Cycle (T<sub>ein</sub>/T):</label>
        <input type="number" id="dutyCycleInput" value ="0.5" step="0.1"><!--value hardgecodet, sollte möglicht der Wert von T sein-->
      </div>
      <div class="input-container">
        <label for="referenceVoltage">Reference Voltage (V):</label>
        <input type="number" id="referenceVoltageInput" value ="5"><!--value hardgecodet, sollte möglicht der Wert von T sein-->
      </div>
    </div>
  </div>
  <script>
  //=======================================Differential Equasion=========================================
    // Parameter
    let V_in = 20; // Eingangsspannung in Volt
    let L = 99e-3; // Induktivität in Henry
    let C = 1e-3; // Kapazität in Farad
    let R_load = 10; // Lastwiderstand in Ohm
    let T = 30; // Schaltperiode in dt
    let D = 0.5; //DutyCycle = T_ein/T
    let V_ref = 5;// Referenzspannung in Volt
    const dt = 1e-4; // Zeitschritt
    const groundVlt = 1; // In Volts (only for animation purposes, not simulation)
```

```
// Initialbedingungen
let i_L = 0; // Anfangsstrom durch die Induktivität
let i_C = 0; //Kondensatorstrom
let v_C = 0; // Anfangsspannung über den Kondensator
let q = 0; // Initialer Schaltzustand (offen)
let pwmCounter = 0; //Zähler, um im automaticMode eine Periodendauer abzuzählen
let stopSimulation = false; // Flag to stop the simulation
let breakSimulation = false; // Flag to break the simulation
let automaticMode = true; // Automatic mode flag
let startSimulation = false;
//let restartSimulation =false;

// Canvas setup
const plot_canvas = document.getElementById('plot');
const ctxPlot = plot_canvas.getContext('2d');

let t_total = 0; // Gesamtzeit
const iL_data = [];
const vC_data = [];
const q_data = [];
//}

//----------------------------Input-----------------------------------------------------

//Funktion, um mit verschiedenen Tastatur-Eingaben Steuerungen der Simulation vorzunehmen
document.addEventListener('keydown', (event) => {
  if (event.key === 'q') {
    q = 1 - q; // Schaltzustand wechseln
    startSimulation = true;
  } else if (event.key === 's') {
```

```
        stopSimulation = true; // Stoppen der Simulation
    } else if (event.key === 'b') {
        breakSimulation = !breakSimulation; // Pausieren und Fortsetzen der Simulation
    } else if(event.key === 'r'){
        restartSimulation();
    }
});

// Set up event listeners for input changes
document.getElementById('modeButton').addEventListener('click', function() {
    automaticMode = !automaticMode;
    this.textContent = automaticMode ? 'Switch to ManualMode' : 'Switch to AutomaticMode';
    this.classList.toggle('manual', !automaticMode);
    //toggleButton.classList.toggle('manual'); //sorgt dafür, dass der toggleButton im maual-Mode auch grau wird
});
document.getElementById('toggleButton').addEventListener('click', function() {
    q = 1-q;
    startSimulation =true;
});
document.getElementById('restartButton').addEventListener('click', function() {
    restartSimulation();
});
document.getElementById('breakButton').addEventListener('click', function() {
    breakSimulation = !breakSimulation;
    this.textContent = breakSimulation ? 'Run' : 'Break'; //ändert die Textanzeige des breakButtons
    this.classList.toggle('run', breakSimulation); //ruft Befehl zum Farbwechsel des breakButtons auf
});
document.getElementById('resistanceInput').addEventListener('input', function(e) {
    R_load = parseFloat(e.target.value);//falls R_load kleiner 0 oder keine Zahl ist
    if (R_load <= 0 || isNaN(R_load)) {
```

```
      R_load = 0.1;
      e.target.value = R_load;// Setze das Eingabefeld auf 0.1
    }
});
document.getElementById('voltageInput').addEventListener('input', function(e) {
  V_in = parseFloat(e.target.value);
  if (V_in<0 || isNaN(V_in)){
    V_in = 0;
    e.target.value = V_in;
  }
});
document.getElementById('inductanceInput').addEventListener('input', function(e) {
  L = 0.001*parseFloat(e.target.value);
  if (L< 0.001 || isNaN(L)) {
    L=0.001;
    e.target.value = L;
  }
});
document.getElementById('capacitanceInput').addEventListener('input', function(e) {
  C = 0.001*parseFloat(e.target.value);
  if (C< 0.001|| isNaN(C)) {
    C=0.001;
    e.target.value = C;
  }
});
document.getElementById('periodDurationInput').addEventListener('input', function(e) {
  T = Math.round(100000*parseFloat(e.target.value)*dt); //input in ms
  console.log('T:', T);
  if (T<1 ||isNaN(T)) {
    T=1;
    e.target.value =T;
  }
```

```
});
document.getElementById('dutyCycleInput').addEventListener('input', function(e) {
  D =parseFloat(e.target.value);
  if (D<0 ||isNaN(D)) {
    D=0;
    e.target.value =D;
  }
  if(D>1){
    D=1;
    e.target.value =D;
  }
});
document.getElementById('referenceVoltageInput').addEventListener('click', function() {
  V_ref = parseFloat(e.target.value);
  if(V_ref<0 || isNaN(V_ref)){
    V_ref = 0;
    e.target.value = V_ref;
  }
});
//--------------------Funktionen zur Mainpulation der Anzeigen--------------------------


//Ändert den Text des ToggleButton (gekoppelt mit dem Switch-Schaltymbol, s.u.)
function updateToggleButtonText() {
  toggleButton.textContent = q === 0 ? 'SwitchOn' : 'SwitchOff';
}


//Funktion zum Zurücksetzen aller Einstellungen und Anzeigen auf den Anfangszustand
(außer modeButton (bzw. AutomaticMode) und breakButton (bzw. breakSimulation))
function restartSimulation(){
  V_in = 20; // Eingangsspannung in Volt
  L = 99e-3; // Induktivität in Henry
  C = 1e-3; // Kapazität in Farad
```

```
        R_load = 10; // Lastwiderstand in Ohm
        V_ref = 5; //Referenzspannung in Volt
        T = 30; // Schaltperiode in dt
        D = 0.5; //DutyCycle = T_ein/T
        t_total = 0; // Gesamtzeit

        i_L = 0; // Anfangsstrom durch die Induktivität
        i_C = 0; //Kondensatorstrom
        v_C = 0; // Anfangsspannung über den Kondensator
        q = 0; // Initialer Schaltzustand (geschlossen)
        stopSimulation = false; // Flag to stop the simulation
        pwmCounter = 0; //Zähler, um im automaticMode eine Periodendauer abzuzählen
        startSimulation = false;

        // Anzeigen der Eingabefelder zurücksetzen
        //document.getElementById('modeButton').classList.toggle('manual');
        document.getElementById('voltageInput').value = V_in;
        document.getElementById('resistanceInput').value = R_load;
        document.getElementById('inductanceInput').value = L*1000;// Anzeige in mH
        document.getElementById('capacitanceInput').value = C*1000;//Anzeige in mF
        document.getElementById('periodDurationInput').value = T*0.1;
        document.getElementById('dutyCycleInput').value = D;
        document.getElementById('referenceVoltageInput').value = V_ref;

        //Bisherige Graphen löschen
        ctxPlot.clearRect(0, 0, plot_canvas.width, plot_canvas.height); //löscht alle Anzeigen aus dem Plot, auch Text usw.
        iL_data.splice(0, iL_data.length);
        q_data.splice(0, q_data.length);
        vC_data.splice(0, vC_data.length);
    }
```

```
//----------------------------Plot-----------------------------------------------------
function draw_plot() {
  const n = 1000; //amount of data beeing displayed
  V_ref = document.getElementById('referenceVoltageInput').value;
  const showReference =true;
  ctxPlot.clearRect(0, 0, plot_canvas.width, plot_canvas.height);

  const margin = 30; //damit die Plots nicht direkt am Rand des Canvas liegen
  const margin_left = 50;
  const graphWidth = plot_canvas.width - 2 * margin_left;
  const graphHeight = (plot_canvas.height - 4 * margin) / 3;

  // Helper function to plot a graph
  //data: Array der zu plottenden Daten, title, yLabel, yMin, yMax: Minimum und Maximum der y-Achse, um die Skalierung zu definieren, yOffset: Der vertikale Versatz, der angibt, wo der Graph auf dem Canvas beginnen soll
  function plotGraph(data, title, yLabel, yMin, yMax, yOffset, showReference=false, refValue =0) {
    ctxPlot.beginPath();
    ctxPlot.moveTo(margin_left, plot_canvas.height - margin - yOffset);

    data.forEach((point, index) => {
      const x = margin_left + (graphWidth * index) / (data.length - 1);
      const y = plot_canvas.height - margin - yOffset - ((graphHeight * (point - yMin)) / (yMax - yMin));
      ctxPlot.lineTo(x, y);
    });

    ctxPlot.stroke();

    // Draw labels and title
    ctxPlot.fillText(title, margin, plot_canvas.height - margin - yOffset - graphHeight - 10);
    ctxPlot.fillText(yLabel, 10, plot_canvas.height - margin - yOffset - graphHeight / 2);
```

```
        if (showReference) {
            ctxPlot.strokeStyle = 'red';
            ctxPlot.beginPath();
            const refY = plot_canvas.height - margin - yOffset - ((graphHeight * (refValue - yMin)) / (yMax - yMin));
            ctxPlot.moveTo(margin_left, refY);
            ctxPlot.lineTo(margin_left + graphWidth, refY);
            ctxPlot.stroke();
            ctxPlot.strokeStyle = 'black'; // Reset the stroke style to black for the next graph
        }
    }

    const q_data_slice = q_data.slice(-n); //die letzten n werte aus dem Array
    const iL_data_slice = iL_data.slice(-n);
    const vC_data_slice = vC_data.slice(-n);

    const q_last = q_data[q_data.length - 1].toFixed(0); //der letzte Wert
    const iL_last = iL_data[iL_data.length - 1].toFixed(2); //gibt den letzten Wert aus vC_Data auf zwei Stellen gerundet zurück
    const vC_last = vC_data[vC_data.length - 1].toFixed(2);
    const vC_min = Math.min(...vC_data, V_ref);
    const vC_max = Math.max(...vC_data, V_ref);

    ctxPlot.fillText(` t = ${t_total.toFixed(3)}s ` , margin, plot_canvas.height - 20);

    plotGraph(q_data_slice, 'Schaltzustand q(t)', ` ${q_last}` , 0, 1, 0);
    plotGraph(iL_data_slice, 'Spulenstrom i_L(t)', ` ${iL_last}A` , Math.min(...iL_data), Math.max(...iL_data), graphHeight + margin);
    plotGraph(vC_data_slice, 'Ausgangsspannung = Kondensatorspannung v_C(t)', ` ${vC_last}V` , vC_min, vC_max, 2 * (graphHeight + margin), showReference, V_ref);

}
```

```
//------------------------solving differential-eqasion---------------------------------------
function simulate() {
  if (stopSimulation) return;
    if (!breakSimulation){
    if (automaticMode) { //switches q automatically
      if(pwmCounter < (D*T)){q=1;}
      if((D*T) < pwmCounter && pwmCounter < T){q=0;}
      if(pwmCounter===T){pwmCounter=0;}
      pwmCounter = pwmCounter+1;
    }

    const di_L_dt = (V_in * q - v_C) / L;
    const dv_C_dt = (i_L - v_C / R_load) / C;
    // Euler-Verfahren zur Lösung der DGLs
    i_L += di_L_dt * dt;
    v_C += dv_C_dt * dt;
    if (i_L < 0){ //spulenstrom kann nicht kleiner als 0 werden, wegen Diode
      i_L = 0;
    }
    i_C = C*dv_C_dt

    // Gesamtzeit aktualisieren
    t_total += dt;

    // Daten speichern
    iL_data.push(i_L);
    vC_data.push(v_C);
    q_data.push(q);
  }
}
```

```
  //=====================================CirquitAnimation=======================
================
    // Initialize canvas and context
    const cirquit_canvas = document.getElementById('cirquit_canvas');
    const ctxCirquit = cirquit_canvas.getContext('2d');

    //-----------------------------------FUnctions for drawing wires and components----------------
    class Wire{
      constructor(startX, startY, endX, endY,knots){
        this.startX = startX;
        this.startY = startY;
        this.endX = endX;
        this.endY = endY;
        this.knots = knots;
        //global values:
        this.maxVoltage = V_in;
        this.minVoltage = groundVlt;
      }

      adpveSpacing = false;

      // Function to draw a wire
      drawWire() { //black rechangle
        ctxCirquit.beginPath();
        ctxCirquit.lineWidth = 5;
        ctxCirquit.moveTo(this.startX, this.startY);
        ctxCirquit.lineTo(this.endX, this.endY);
        ctxCirquit.strokeStyle = 'black';
        ctxCirquit.stroke();
      }
      getDotColor(value, min, max) {
        // Ensure the value is within the range
```

```
    if (value < min) value = min;
    if (value > max) value = max;

    // Calculate the percentage of the value within the range
    let percentage = ((value - min) / (max - min));

    // Convert the percentage to an RGB color between red and yellow
    // Red is (255, 0, 0) and Yellow is (255, 255, 0)
    let r = 255; // Red is always 255
    let g = Math.round(255 * (1-percentage));
    let b = 0;

    // Convert RGB to hex
    let hex = "#" + ((1 << 24) + (r << 16) + (g << 8) + b).toString(16).slice(1).toUpperCase();
    return hex;
}

// Function to draw the moving dots along a wire
drawDots = function () {
    let dotRadius = 3;
    let integralCurrent = 0; //permanent value
    return function (current, voltage) { //used to make values permanent
        let dotSpacing = 50;
        let dotColor = this.getDotColor(voltage, this.minVoltage, this.maxVoltage);
        integralCurrent += current;
        if (this.adpveSpacing){
            dotSpacing = 50/voltage;
            integralCurrent = integralCurrent/voltage;
        }
        const dotSpeed = integralCurrent; // Speed of the dots, proportional to current
        const time = Date.now() / 1000; // Get current time in seconds
```

```
      const dx = this.endX - this.startX;
      const dy = this.endY - this.startY;
      const distance = Math.sqrt(dx * dx + dy * dy);
      const angle = Math.atan2(dy, dx);

      for (let d = (dotSpeed) % dotSpacing; d < distance; d += dotSpacing) {

        const x = this.startX + Math.cos(angle) * d;
        const y = this.startY + Math.sin(angle) * d;
        if (d>0.1){
          ctxCirquit.beginPath();
          ctxCirquit.arc(x, y, dotRadius, 0, 2 * Math.PI);
          ctxCirquit.fillStyle = dotColor;
          ctxCirquit.fill();
        }
      }
    }
}()
drawKnot(x, y) { //knot at each end of a wire
    const knotRadius = 5;
    ctxCirquit.beginPath();
    ctxCirquit.arc(x, y, knotRadius, 0, 2 * Math.PI);
    ctxCirquit.fillStyle = 'black';
    ctxCirquit.fill();
}
// Function to create a wire with moving dots
createWire(current, voltage) {
    this.drawWire();
    this.drawDots(current, voltage);
    if (this.knots){
      this.drawKnot(this.startX, this.startY);
      this.drawKnot(this.endX, this.endY);
```

```
    }
  }
}

class Component extends Wire {
  constructor(startX, startY){
    super(); //calls parent
    this.startX = startX;
    this.startY = startY;
    this.knots = false;
  }

  drawWire_(startX,startY,endX,endY) {
    ctxCirquit.beginPath();
    ctxCirquit.lineWidth = 5;
    ctxCirquit.moveTo(startX, startY);
    ctxCirquit.lineTo(endX, endY);
    ctxCirquit.strokeStyle = 'black';
    ctxCirquit.stroke();
  }
  createDiode() {
    let startX = this.startX;
    let startY = this.startY;
    this.drawWire_(startX-25, startY, startX+25, startY);
    this.drawWire_(startX-25, startY+50, startX, startY);
    this.drawWire_(startX+25, startY+50, startX, startY);
    this.drawWire_(startX-25, startY+48, startX+25, startY+48);
  }
  drawMagneticField(startX, startY, endX, endY, scalingFactor) {
    ctxCirquit.beginPath();
    ctxCirquit.lineWidth = 1;
    ctxCirquit.strokeStyle = 'blue';
```

```
    const loops = 5; // Number of loops in the inductor
    const totalLength = endX - startX;
    const loopSpacing = totalLength / (loops * 2); // Space for each half loop

    let currentX = startX;
    scalingFactor = scalingFactor*2

    for (let i = 0; i < loops; i++) {
      // Draw magnetic field lines around each loop
      for (let j = 1; j <= scalingFactor; j++) {
        const offset = j * loopSpacing / (scalingFactor + 1);
        ctxCirquit.moveTo(currentX + offset, startY - loopSpacing);
        ctxCirquit.bezierCurveTo(
          currentX + offset - loopSpacing, startY - loopSpacing * 2,
          currentX + offset + loopSpacing, startY - loopSpacing * 2,
          currentX + offset + loopSpacing, startY - loopSpacing
        );
      }
      currentX += loopSpacing * 2;
    }

    ctxCirquit.stroke();
  }

  drawInductor(endX, endY, current) {
    let startX = this.startX;
    let startY = this.startY;
    ctxCirquit.beginPath();
    ctxCirquit.lineWidth = 5;
    ctxCirquit.strokeStyle = 'black';
```

```
    const loops = 5; // Number of loops in the inductor
    const totalLength = endX - startX;
    const loopSpacing = totalLength / (loops * 2); // Space for each half loop

    let currentX = startX;
    for (let i = 0; i < loops; i++) {
      // Draw the uper half of the loop
      ctxCirquit.arc(currentX + loopSpacing, startY, loopSpacing, Math.PI, 0, false);
      currentX += loopSpacing * 2;

    }

    ctxCirquit.stroke();
    this.drawMagneticField(startX, startY, endX, endY, current);
  }

  createSwitch(state, current, voltage){
    let startX = this.startX;
    let startY = this.startY;
    let shiftY = 0;
    let shiftX = 0;
    let knots = false;
    if(state){
      shiftY = 0;
      shiftX = 0;
      knots = true;
      updateToggleButtonText();
    }
    else{
      shiftY = 30;
      shiftX = 5;
      knots = false;
```

```
      updateToggleButtonText();
    }
    this.endX = startX+50-shiftX;
    this.endY = startY-shiftY;
    this.knots = knots;
    this.createWire(current, voltage);

}

createCapacitor(current, highVoltage, lowVoltage){
    let startX = this.startX;
    let startY = this.startY;

    let upLeft = new Wire(startX-25,startY,startX,startY, false)
    upLeft.adpveSpacing = true;
    upLeft.createWire(current,highVoltage);
    let upRight = new Wire(startX+25,startY,startX,startY, false)
    upRight.adpveSpacing = true;
    upRight.createWire(current,highVoltage);
    let downLeft = new Wire(startX-25,startY+50,startX,startY+50, false)
    downLeft.adpveSpacing = true;
    downLeft.createWire(current,lowVoltage);
    let downRight = new Wire(startX+25,startY+50,startX,startY+50, false)
    downRight.adpveSpacing = true;
    downRight.createWire(current,lowVoltage);
}
createResistor(){
    let startX = this.startX;
    let startY = this.startY;
    this.drawWire_(startX-18,startY,startX+18,startY); //up
    this.drawWire_(startX+15,startY,startX+15,startY+70, false); //right
    this.drawWire_(startX-15,startY,startX-15,startY+70, false); //left
```

```
    this.drawWire_(startX-18,startY+70,startX+18,startY+70, false); //down
  }
  createVoltSrc(){
    const radius = 30;
    let startX = this.startX;
    let startY = this.startY+radius;

    ctxCirquit.beginPath();
    ctxCirquit.arc(startX, startY, radius, 0, 2 * Math.PI);
    ctxCirquit.fillStyle = 'black';
    ctxCirquit.stroke();
  }
  drawArrow(startX,startY,endX,endY) {
    ctxCirquit.beginPath();
    ctxCirquit.lineWidth = 3;
    ctxCirquit.moveTo(startX, startY);
    ctxCirquit.lineTo(endX, endY);
    ctxCirquit.strokeStyle = 'blue';
    ctxCirquit.stroke();
  }
}
//-----------------------------------placing wires and components---------------------------------
//switch-branch
switch1 = new Component(100,50);
wire1 = new Wire(50, 50, 100, 50, true);
wire2 = new Wire(150, 50, 250, 50, true);
wire17 = new Wire(50, 350, 50, 150, true)
voltSrc = new Component(50,150);
wire3 = new Wire(50, 150, 50, 50, true);
wire4 = new Wire(250, 350, 50, 350, true);
//diode-brach
diode = new Component(250, 150);
```

```
    wire5 = new Wire(250, 150, 250, 50, true);
    wire6 = new Wire(250, 350, 250, 150, true);
    //capacitor-branch
    capacitor = new Component(450, 150);
    wire8 = new Wire(450, 150, 450, 50, false);
    wire15 = new Wire(450,200,450,350,false);
    //inductor-branch (and below)
    inductor = new Component(320, 50);
    wire9 = new Wire(250, 50, 320, 50, true);
    wire10 = new Wire(400, 50, 450, 50, true);
    wire11 = new Wire(450, 50, 550, 50, true);
    //resistor-branch
    resistor = new Component(550, 150);
    wire7 = new Wire(450, 50, 550, 50, true);
    wire12 = new Wire(550, 350, 450, 350, true);
    wire13 = new Wire(550, 50, 550, 150, true);
    wire16 = new Wire(550, 220, 550, 350, true);
    wire14 = new Wire(450, 350, 250, 350, true);
    //arrows: first number: 1: Voltage source, 2: i_L, 3: v_C
    arrow11 = new Component();
    arrow12 = new Component();
    arrow13 = new Component();
    arrow21 = new Component();
    arrow22 = new Component();
    arrow23 = new Component();
    arrow31 = new Component();
    arrow32 = new Component();
    arrow33 = new Component();

    //------------------------assingning voltages and currents to Wires and Components----------------
-------
    function drawCirquit() {
```

```
ctxCirquit.clearRect(0, 0, cirquit_canvas.width, cirquit_canvas.height); // Clear canvas


const conversion_factor = 2;
let inductorCrt = i_L*conversion_factor;
let capacitorCrt = i_C*conversion_factor;
let resisitorCrt = inductorCrt - capacitorCrt;
let capacitorVlt = groundVlt + v_C;
let sourceVlt = groundVlt+ V_in;


//switch-branch
switch1.createSwitch(q, inductorCrt*q, sourceVlt);
wire1.createWire(inductorCrt*q, sourceVlt);
wire2.createWire(inductorCrt*q, sourceVlt*q + groundVlt*(1-q));
wire17.createWire(inductorCrt*q, groundVlt);
voltSrc.createVoltSrc();
wire3.createWire(inductorCrt*q, sourceVlt);
wire4.createWire(inductorCrt*q, groundVlt);
//diode-brach
wire5.createWire(inductorCrt*(1-q), sourceVlt*q + groundVlt*(1-q));
wire6.createWire(inductorCrt*(1-q), groundVlt);
diode.createDiode();
//capacitor-branch
wire8.createWire(-capacitorCrt, capacitorVlt);
wire15.createWire(capacitorCrt, groundVlt);
capacitor.createCapacitor(capacitorCrt, capacitorVlt, groundVlt);
//inductor-branch (and below)
wire9.createWire(inductorCrt, sourceVlt*q + groundVlt*(1-q));
wire10.createWire(inductorCrt, capacitorVlt);
wire14.createWire(inductorCrt, groundVlt);
inductor.drawInductor(400, 50, inductorCrt);
//resistor-branch
resistor.createResistor();
```

```
        wire7.createWire(resisitorCrt, capacitorVlt);
        wire12.createWire(resisitorCrt, groundVlt);
        wire16.createWire(resisitorCrt, groundVlt);
        wire13.createWire(resisitorCrt, capacitorVlt);
        //arrows: first number: 1: Voltage source, 2: i_L, 3: v_C
        ctxCirquit.font = "20px Arial"; //bestimmt Schriftgröße und -art
        ctxCirquit.fillStyle = "blue";  //Schriftfarbe
        arrow11.drawArrow(10,115,10,250);
        arrow12.drawArrow(3,243,10,250);
        arrow13.drawArrow(10,250,17,243);
        ctxCirquit.fillText("V_in", 5, 280);
        arrow21.drawArrow(300,25,420,25);
        arrow22.drawArrow(413,18,420,25);
        arrow23.drawArrow(420,25,413,32);
        ctxCirquit.fillText("i_L", 430, 30);
        arrow31.drawArrow(500,115,500,250);
        arrow32.drawArrow(493,243,500,250);
        arrow33.drawArrow(500,250,507,243);
        ctxCirquit.fillText("v_C", 480, 280);

        ctxCirquit.fillText("q(t)", 110, 75);
      }
   //============================update simaulation, plot and
animation============================
      function update() {
        if(!startSimulation){
          drawCirquit();
        }
        if(startSimulation&&!breakSimulation){
          simulate();
          draw_plot();
          drawCirquit();
```

```
        }
        requestAnimationFrame(update); //funktion, die von Javascript bereit gestellt wird zum neuladen
    }
    update(); //wird permanent aufgerufen
  </script>
</body>
</html>
```