Finite state machines - Traffic Light

Look in the samples folder for the script If you don't want to type it in..
`https://editor.p5js.org/spikol/sketches/wM_mzde00`

```
// States for our FSM (Finite State Machine)
const STATE_RED = "RED";
const STATE_GREEN = "GREEN";


// Initial state
current_state = STATE_RED;

function setup() {
 // Create a canvas with a width and height of 200 pixels
  createCanvas(200, 200);
  fill(255);  // Set the default fill color to white
}

function draw() {
  background(200);  // Set the background color to a light gray

  // Check the current state and set the fill color accordingly
  if (current_state === STATE_RED) {
    fill(255, 0, 0);  // Red color for RED state
  } else if (current_state === STATE_GREEN) {
    fill(0, 255, 0);  // Green color for GREEN state
  }
  // Draw a circle in the center of the canvas
  ellipse(width / 2, height / 2, 100, 100);
}

function mousePressed() {
  // Update the state based on the current state
  if (current_state === STATE_RED) {
    current_state = STATE_GREEN;
  } else if (current_state === STATE_GREEN) {
    current_state = STATE_RED;
  }
}
```


DIKU
ds@di.ku.dk

Details- Traffic Light

1. **State Constants:**
   - The state variables `STATE_RED`, `STATE_GREEN` are directly converted to JavaScript constants using `const`.
2. **Initial State:**
   - The initial state is set using `current_state = STATE_RED;` to indicate the starting state.
3. **setup() Function:**
   - `createCanvas(200, 200)` is used to set up a 200x200 pixel canvas.
   - `fill(255)` sets the initial fill color to white, though this will be overridden in `draw()` based on the state.
4. **draw() Function:**
   - The `draw()` function is called repeatedly to render the scene.
   - It checks the current state (`current_state`) and sets the fill color accordingly (red or greeen).
   - A circle is drawn in the center of the canvas using the `ellipse()` function.
5. **mousePressed() Function:**
   - The `mousePressed()` function handles the state transitions when the mouse is pressed.
   - It cycles through the states (`STATE_RED` and `STATE_GREEN` in the order specified.

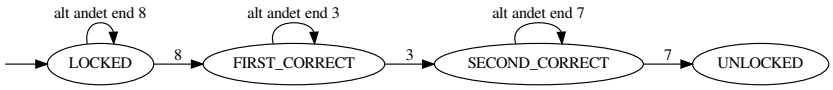
Finite state machines - Task

Your mission is as follows: the traffic light self-destruct.

- Add `STATE_YELLOW`
- Remember to add to `const`
- Also you will need to modify the `function draw()` and `function mousePressed()` for state and the trigger.

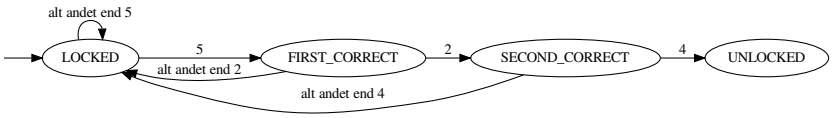
Finite state machines - Padlock

As the first example of a state machine, we will create one electronic combination lock, such as those used for access control doors.
Tilstandsdiagram:



Copy the p5js code for the combination from the shared collection:
`https://editor.p5js.org/spikol/sketches/qqyKLj46m`

- Test the combination lock and watch as the state changes. If you did not know the ransom (password), how many attempts does it require? to guess?
- Change the code so that the default is 524 instead
- Try changing the code to follow this diagram instead, where incorrect presses reset the state to start:

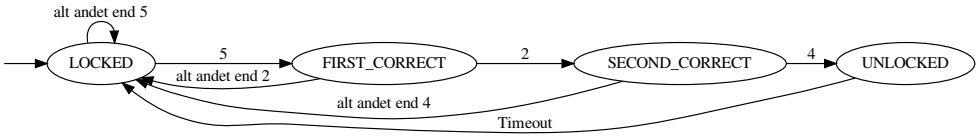


- Draw an extended state diagram of the combination lock with an extra mode, so 4 digits are required. Next, add the extra state in the code.

DIKU
ds@di.ku.dk

Automatic relock after 2 seconds

Let's extend the lock so that the door automatically locks again after 2 seconds, which corresponds to 120 frames:



- Create a global variable "`timer`" and set it to 0
- Set the `timer` variable to 120 as soon as the lock is unlocked. It will say when it changes `lockState` to `"UNLOCKED"`.
- Count down with the timer in each frame (add the following to the `draw` function):

```
if (lockState === "UNLOCKED") {
    drawLock(width / 2, height / 2, 50, true);
    timer -= 1;  // Decrement the timer
```

- The lock must be opened When the timer has counted down. The `draw` function must now check if we are in the mode `"UNLOCKED"` and the timer simultaneously has counted down to 0.
- Paste the following in `draw`:

```
if (timer === 0) {
    lockState = "LOCKED";  // Reset to LOCKED state when timer
        runs out
  }
```

Introduction to Lists and For Loops

With **for** loops, we can avoid duplicated code and iteratively operate, such as drawing 3 circles on a screen. Start a new project and insert the following code:

```
// Initialize Lists

// List of x-coordinates for the circles
xs = [30, 50, 70, 90, 110, 130];
// List of y-coordinates for the circles
ys = [30, 50, 70, 90, 110, 130];

function setup() {
  // Create a canvas with a width and height of 400 pixels
  createCanvas(400, 400);
  // Set the background color to black
  background(0);
}

function drawCircle(x, y) {
  // Disable the outline (stroke) for the circles
  noStroke();
  // Generate a random red color value between 0 and 255
  bubble_R = random(256);
  // Generate a random green color value between 0 and 255
  bubble_G = random(256);
  // Generate a random blue color value between 0 and 255
  bubble_B = random(256);
  // Set the fill color using the random RGB values
  fill(bubble_R, bubble_G, bubble_B);
  // Draw a circle at the specified (x, y) coordinates
  circle(x, y, 25);
}

function draw() {
  // Call the drawCircle function for each pair of (x, y)
     coordinates
  for (var i = 0; i < xs.length; i++) {
    drawCircle(xs[i], ys[i]);
  }
}
```


DIKU
ds@di.ku.dk

Explanation Loops I

What happens when you run the code? Explain here how a loop produces the result, and give yourself time to understand the logic. This code draws multiple circles on a black background at predefined positions. Each circle is filled with a random color, and the process repeats every frame (due to the `draw()` function), although the positions and colors remain constant unless the `xs` or `ys` arrays are changed dynamically during execution.

1. **Global Variables:**
   - `xs` and `ys` are arrays (lists) that store the x and y coordinates for the circles. Each pair of values from these arrays will be used to draw a circle on the canvas.
2. **setup() Function:**
   - The `setup()` function is called once when the program starts. It creates a canvas of 400x400 pixels and sets the background color to black.
3. **drawCircle(x, y) Function:**
   - This is a custom function designed to draw a single circle at the specified (x, y) coordinates.
   - Inside this function:
     - `noStroke()` is called to ensure the circles don't have an outline.
     - `random(256)` is used to generate random values for the red (`bubble_R`), green (`bubble_G`), and blue (`bubble_B`) color components, resulting in a random color.
     - `fill(bubble_R, bubble_G, bubble_B)` sets the fill color for the circle using the randomly generated RGB values.
     - `circle(x, y, 25)` draws a circle at the specified coordinates with a diameter of 25 pixels.
4. **draw() Function:**
   - The `draw()` function is called repeatedly, continuously executing the code within it.
   - Inside `draw()`, a `for` loop iterates over the `xs` array. For each iteration, it retrieves the corresponding x and y coordinates from `xs` and `ys` arrays and passes them to the `drawCircle()` function to draw a circle.
   - As a result, a set of circles is drawn on the canvas at the specified coordinates, each with a random color.

DIKU
ds@di.ku.dk

More with Lists and For Loops II

The following code snippet has a few changes: We now use a loop to generate our list! Paste the code into a new project and consider the differences in the various loops.

```
// Initialize Lists
// Empty array to hold x-coordinates for the circles
circleList = [];

function setup() {
  createCanvas(400, 400);
  background(0);
}

function drawCircle(x, y) {
  noStroke();
  // Generate a random  colors value between 0 and 255
  bubble_R = random(255);
  bubble_G = random(255);
  bubble_B = random(255);
   // Set the fill color using the random RGB values
  fill(bubble_R, bubble_G, bubble_B);
  circle(x, 200, 50);  // Draw a circle
}

function createCircleList(n) {
// Add a new x-coordinate to the circleList array, spaced by 60
    pixels
  for (var i = 0; i < n; i++) {
    append(circleList, i * 60);    }
}

function draw() {
// Draw a circle at each x-coordinate in the circleList, with y
    fixed at 200
  createCircleList(10);  // Populate the circleList with 10
      x-coordinates
  for (var i = 0; i < circleList.length; i++) {
    drawCircle(circleList[i], 200);
  }
}
```


DIKU
ds@di.ku.dk

Explanation for Loops II

- **createCircleList(n) Function:**
  - A loop runs n times, appending new values to circleList.
  - Each value added is i * 60, where i is the loop counter, resulting in x-coordinates spaced 60 pixels apart.
- **draw() Function:**
  - createCircleList(10): Calls the createCircleList() function to generate 10 x-coordinates and store them in circleList.
  - A loop then iterates over circleList, calling drawCircle() for each x-coordinate, with the y-coordinate fixed at 200 pixels.
  - This results in 10 circles being drawn on the canvas.

Debugging Strategies

Essential Skills for Every Coder *Your best debugging tool is something you already have: your brain!*

Getting errors is totally normal - it happens to every programmer, every time they write code.

Start with Your Brain

Be Specific About the Problem

- Don't say: "I wanted to draw a flower but it's not working"
- Do say: "I expected a red circle in the upper-left corner, but I only see half a circle at the top"

Read Your Code Line by Line

- Double-check spelling, capitalization, keywords, and symbols
- Use paper and pencil to trace through your code
- Write down variable values as you go
- Don't assume - take it one line at a time
- Reading out loud can help!

Three Types of Errors

1. **Syntax Errors** - The Computer can't understand your code

```
function draw {  // Missing () parentheses!
  background(220);
}
// Error: Unexpected token '{'
```

2. **Runtime Errors** - Code runs but breaks during execution.

```
function draw() {
  background(myColor);  // myColor is not defined!
}
// Error: myColor is not defined
```

3. **Logic Errors** - Code runs but does something unexpected

```
// Circle appears in wrong place because width/height
// are calculated before createCanvas()
```

Research Errors Effectively

- *Copy the exact error message* and search for it
- Add "p5.js" or "JavaScript" to your search
- Use quotes for exact matches: ‘"Unexpected token"‘
- Look for Stack Overflow and Processing forum posts
- Sometimes search only part of the error (without your variable names)
- Use AI with an active learning approach

Best Debugging Practices

Start Small and Test Often

1. Don't write your whole program at once!
2. Test after every few lines of code
3. It's easier to find bugs in code you just wrote
4. Create small example programs (MCVE*) to isolate problems

**Minimal Complete Verifiable Example*

Other Powerful Techniques

Start Small and Test Often

- **Rubber Duck Debugging** - Explain your problem out loud to anything
- Take Breaks - Solutions often come when you step away
- Comment Out Code - Use // to temporarily disable problematic sections
- Use Browser Developer Tools - Press F12 for advanced debugging
- Use AI support tools with active engagement