Welcome to Kickstart

Welcome to Computer Science and the programming kickstart course! The exercises below will quickly get you up and running with programming P5 JavaScript. We have intentionally left some things unexplained as we want you to experiment on your own.

Resources

There is a world of information and resources to help you learn p5.js. Check out the following:

Book McCarthy, Lauren, Casey Reas, and Ben Fry. Getting started with P5. JS: Making interactive graphics in JavaScript and Processing. Maker Media, Inc., 2015.

p5.js Web site The official website

Happy Coding A fun set of tutorials and examples for p5.js: Happy Coding

Kickstart Absalon You can find the book and other materials on the LMS.

Have Fun

DIKU
ds@di.ku.dk

First Program

Type the following example in the Processing editor and press -button:

```
function setup() {
  //set the canvas and background color
  createCanvas(400, 400);
  background(220);

  // Tree
  rect(55, 50, 10, 20);
  ellipse(60, 35, 30, 40);
}

function draw() {}
```

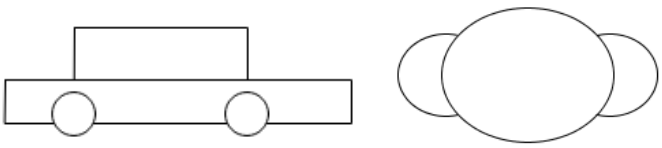
Then add the following into the `setup()` function:

```
  // powerplant
  rect(120, 50, 60, 30);
  rect(160, 20, 10, 30);
  triangle(120, 50, 136, 40, 136, 50);
  triangle(136, 50, 152, 40, 152, 50);
```

And:

```
  // windmill
  line(300, 50, 320, 51);
  line(300, 50, 289, 67);
  line(300, 50, 291, 32);
  line(300, 50, 300, 90);
```

Now try drawing a car and a cloud (psst, use the documentation):



Documentation

Documentation is a manual describing how a specific part of a programming language works. This is an easy way to learn what all the numbers refer to in `triangle(120, 50, 136, 40, 136, 50)`.
Use the **documentation** on p5.js. For instance, search for *triangle* and press the reference `triangle()`. Then you can read what each of the parameters (the numbers in paremthesis) given to `triangle()` do.

DIKU
ds@di.ku.dk

Colors

You are now going to colour the shapes. To do this, use the `fill(r, g, b)` function, which selects which colour to use for fill and text colour. It's all about calling `fill` in the right places! As arguments, you specify the amount of red (0-255), blue (0-255), and green (0-255). Here are some basic colors to experiment with:

```
fill(255, 0, 0);  // red
fill(0, 255, 0);  // green
fill(0, 0, 255);  // blue
```

```
fill(0, 0, 0);       // black
fill(255, 255, 255); // white
fill(255, 255, 0);   // yellow
```

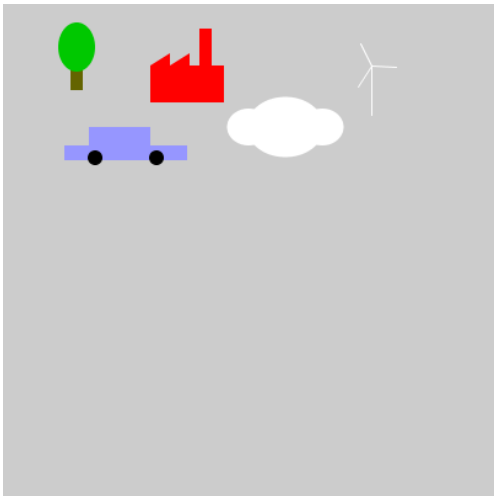
Optionally, find colours using an online color picker or RGB colour table. For example, search for "RGB color picker".

Lines and outlines

To specify the colour of strokes (e.g. `line`) and outlines, use `stroke(r, g, b)`. Also try the `noStroke()` function, to turn off outline drawing.

Example

Here's an example of what it might look like after colouring:

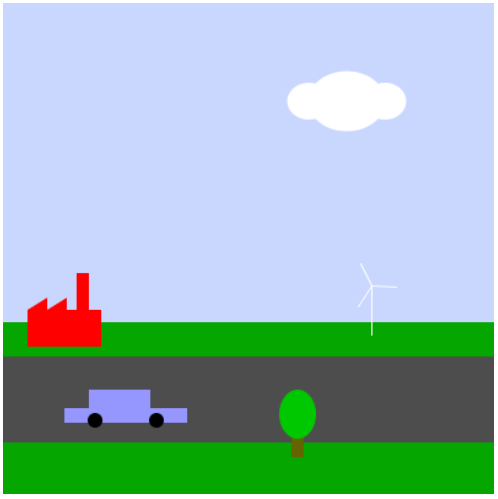


DIKU
ds@di.ku.dk

Green City

Use what you've learned to make it a slightly nicer scene, with background, foreground and the extra details you think should be there. For example, I've drawn a road and moved the objects around.

Tip: To change the background color from gray you can use `rect` which you already know, but you can also use the `background(r, g, b)` command. It will delete everything and fill the screen with the specified colour.



Comments

A comment is any line starting with `//`. The line won't be interpreted as code by the program. A comment can be used to "remove" a piece of code temporarily for debugging or for helping yourself understand the code with an easy to understand label, for example:

```
// the next line is a comment, but the last line is code
// fill(255, 0, 0);
fill(255, 0, 0);
```


DIKU
ds@di.ku.dk

Making an Aquarium

Create a new project ("File" -> "New") and immediately save it the project. Call it "Aquarium".

Type in this piece of code:

```
function setup() {
  createCanvas(400, 400);
  background(220);

  //basic fish shape
  fishX = 150;
  ellipse(fishX, 200, 120, 75);
  triangle(fishX - 60, 200, fishX - 90, 170, fishX - 90, 230);
}

function draw() {

}
```

Try changing 150 to a different number in the `fishX` specification.
Now add the following:

```
  eyeSize = 15;
  ellipse(fishX + 30, 190, eyeSize, eyeSize);
```

Try changing the value of `eyeSize`.

Variables

A variable is a piece of data linked to a name. This could be a number, a piece of text, or one of the many other data types.

It is defined with the following syntax:

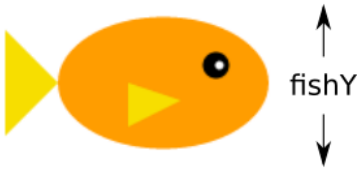
```
x = 1;
y = 2;
z = x + y;
```


DIKU
ds@di.ku.dk

Tasks

You have now added a fish that can be moved just by changing one value.

- Color the fish
- Give the fish a fin that moves when you change `fishX`.
- Give the fish a pupil that moves when you change `fishX`.
- Create a new variable, `fishY`, that controls the y-position of the fish



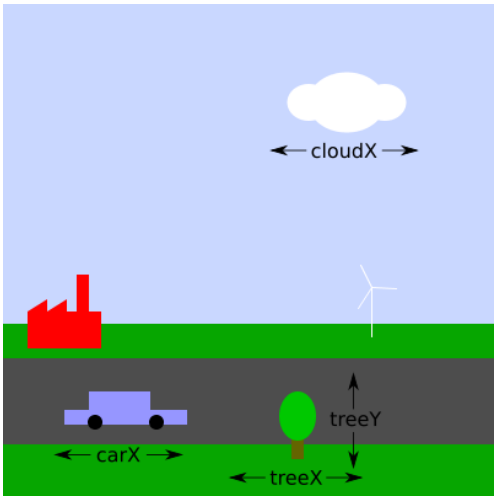
Remember to save the project. We'll be working on it later.

DIKU
ds@di.ku.dk

Green City continues

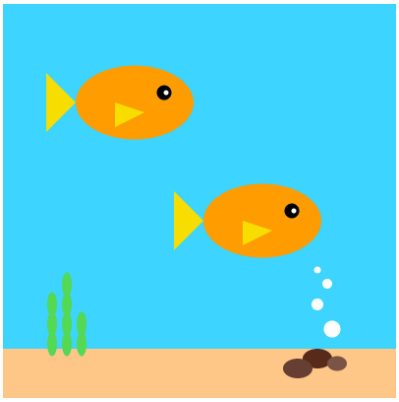
Switch to the electric car project and introduce variables to specify the location of the objects so that we can later animate these objects.

- Create a variable `carX` so the car can move forward and back
- Create a variable `cloudX` so the cloud can move back and forth
- Create a variable `treeX` so the tree can be moved horizontally
- Create a variable `treeY` so the tree can be moved vertically



Aquarium Continues

Use what you've learned to expand your aquarium project; here's an example, but feel free to use your imagination! For example, one variable is used for the x-coordinate of the seaweed plant and another variable for the x-coordinate of the entire group of rocks (as a unit). An additional set of variables `fish2X`/`fish2Y` to control the location of the additional fish.



Functions

Functions allow you to reuse the same code in multiple places, name entire blocks, and add structure to code.

Open the Aquarium project. Add the following function to draw a fish:

```
function setup() {
  // Set up the canvas
  createCanvas(400, 400);

  // Set the background color to light grey
  background(220);
}

function draw() {
  // Draw the first fish at x-coordinate 100
  drawFish(100);

  // Draw the second fish at x-coordinate 280
  drawFish(280);
}

function drawFish(fishX) {
  // Draw the basic fish shape
  // Draw fish body  (fishX, 200) with a width of 120 and height of
      75
  ellipse(fishX, 200, 120, 75);

  // Draw the fish tail using a triangle
  // The triangle points are at (fishX - 60, 200), (fishX - 90,
      170),
  //and (fishX - 90, 230)
  triangle(fishX - 60, 200, fishX - 90, 170, fishX - 90, 230);

  // Draw the fish eye
  // The eye is a small ellipse located at (fishX + 30, 190)
  //with a size of 15x15
  eyeSize = 15;
  ellipse(fishX + 30, 190, eyeSize, eyeSize);
}
```

Now it's much faster to fill the aquarium with fish and we avoid copying code.

Functions

A function in programming is like a function in math. It takes one or more arguments in like x, which could be a number (Int) type. It then utilizes the input to do or compute something, which can be returned as a result.

An example of a function that returns an output is `plus2`. `plus2` takes a number and returns the number plus 2:

```
function plus2(a){
    return a + 2
}
```

In the example above, `a` is an input/parameter to the function that needs to be a number when calling the function. The function returns a number that can be used or bound elsewhere in the code.

A function does not need to return anything in p5.js. An example of this is the drawFish function above, which draws the fish without any returns.

Task

Create your own `drawFish(x, y)` function that draws your entire fish with colour, fins and eyes.

Notes and Extra

- Try changing 50 to another number
- Try changing the line `x = x + 1` to `x = x - 1` or to `x = x + 5`
- Try moving the call to `background` from `draw` to `setup` - what happens?

BE AWARE

When using setup/draw, call draw functions outside of setup and draw are not allowed. Everything must be moved into the two functions.

DIKU
ds@di.ku.dk

Green City continued

Over in the electric car project, you can also try writing a function to draw trees:

```
 function setup() {
  createCanvas(400, 400);
  background(220);
  drawTree(160);
}

function draw() {
  // we are going to use the draw function soon!
}

function drawTree(treeX) {
  fill(100, 100, 0);
  rect(treeX - 5, 350, 10, 20);
  fill(0, 200, 0);
  ellipse(treeX, 335, 40, 50);
}
```

Structure your electric car project code with functions:

- Write a `drawCloud(x)` function that draws a cloud
- Extend the `drawTree(x)` function to also accept a y-coordinate
- Write a `drawCar(x)` function that draws a car
- Write a `drawPowerplant()` function and a `drawWindmill()` function that draws the power plant and the windmill, respectively. We won't need to move them around, so they don't need to not take coordinates as an argument.

Call all the functions in the `setup()` for now. For example:

```
drawTree(150, 235);
drawTree(240, 335);
drawPowerplant();
drawWindmill();
drawCar(50);
drawCloud(280);
```

Animation and Functions

Create a new temporary project (you don't need to save it). Type in this piece of code:

```
// declare global variable accessible from any part of the code.
// global variables are declared outside of any functions.
globalVarX = 50;

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

function draw() {
  background(220);
  background(255, 255, 255);
  fill(255, 0, 0);
  ellipse(globalVarX, 100, 30, 30);
  globalVarX = globalVarX + 1;
}
```

The `draw` function is automatically called 60 times per second!

Tasks

- Try changing 50 in `globalVarX` to different number.
- Try changing the line `x = x + 1` to `x = x - 1` or to `x = x + 5`
- Try moving the call to `background` from `draw` to `setup` - what happens?

NOTE!!:

When using `setup/draw` it is not allowed to call drawing functions outside of `setup` and `draw`. Everything must be moved into the two functions.

Aquarium continues

- *Rewriting the aquarium project to use `setup/draw`:*
  - Add empty `setup` and `draw` functions to the bottom of the program
  - Call `createCanvas(400, 400)` in `setup`
  - Call all the drawing functions in `draw`, incl. drawing of the background
- *Make the fish swim:*
  - Create two global variables `fish1X` and `fish2X` (before `setup/draw`)
  - Use the new variables as x-argument when you call `drawFish()`
  - Update the variables with $+1/-1$ inside the `draw` function

DIKU
ds@di.ku.dk

Green City continues

- Create two global variables `carX` and `cloudX`
- Make the car drive to the right
- Make the cloud start outside the image on the right side and move to the left

Randomness

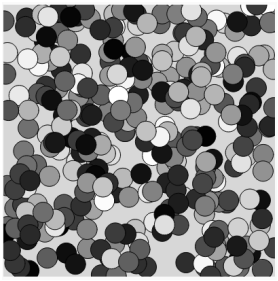
Create a brand new project and save it as "random_circles". Add the following code:

```
function setup() {
  createCanvas(400, 400);
  background(220);
}

function draw() {
  // create 2 variables for different x
      y coordinates on the canvas
  x = random(0, width);
  y = random(0, height);

  // color the circle with a random
      monochrome shade
  fill(random(255));

  // draw the ellipse every frame
  ellipse(x, y, 30, 30);
}
```



Tasks

- Make the circles change size randomly
- Make the circles be drawn in random colours. Experiment until you find a colour scale that you think is nice.

DIKU
ds@di.ku.dk

Interaction with the Mouse

The position of the mouse can be read with the variables `mouseX` and `mouseY`. A kind of drawing program can be written like this:

```
function setup() {
  createCanvas(800, 800);
  background(255, 204, 0);
}

function draw() {
  fill(0, 0, 0);
  ellipse(mouseX, mouseY, 5, 5);
}
```

Keyboard input

Open the drawing program project and try adding the following new function:

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

function draw() {

}

function keyPressed(){
  background(0,0,255,255);
}
```

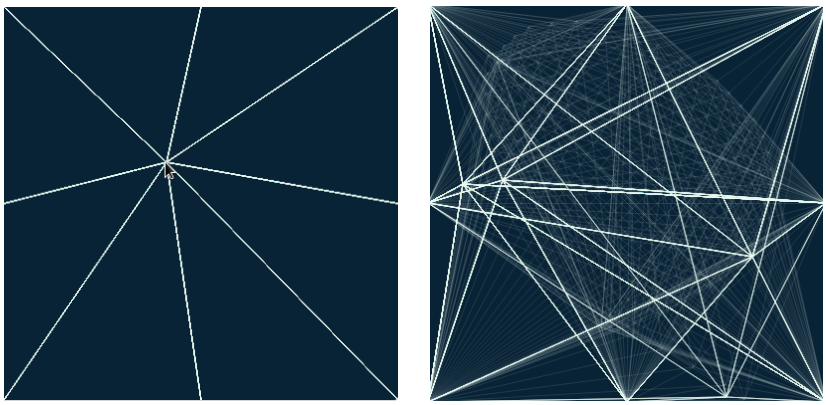
Start the program and press any key on the keyboard. To check for a specific key, the variable "`key`" can be read, like in the following code. It can be added to the `keyPressed()` function.

```
if (key == 'c') { // Check if the 'c' key is pressed
   background(255, 204, 0); // Set background color to a blue hue
```


DIKU
ds@di.ku.dk

New Project:Creative Programming

The task is now to create a more elaborate drawing program. Sign lines from all corners and midpoints on the sides. Use the variables `width` and `height`.



Remember to save the project!

DIKU
ds@di.ku.dk

Introduction: Conditionals

With conditions, we can make things happen when special criteria are met met. Open the aquarium project and try inserting the following into it The `draw` function:

```
if (fish1X > 500) {
    fish1X = -40;
}
if (fish2X < -50) {
fish2X = 200;
}
```

What happens?

Conditionals

In conditionals, we define a condition that is checked each time the code runs. This can be any condition (e.g. $Fish1X \geq 10$) that evaluates to either true or false. When the condition evaluates to true, the code in the brackets runs. If the condition is not met, an alternative outcome can be defined by an else statement. See the following syntax:

```
if (condition) {
  //Code executed if the condition is true
} else {
  //Code executed if the condition is false
}
```

Change direction

By making a variable that contains the direction in which the fish swims, we can change the direction when it reaches the sides.

- Define a global variable `fish1XVelocity` and set it to 1
- Change `fish1X = fish1X + 1` to `fish1X = fish1X + fish1XVelocity`
- Remove the previous `fish1X` condition
- Add these conditions:

```
if (fish1X > 400){
    fish1XVelocity = -1
}
if (fish1X < 0){
    fish1XVelocity = 1
}
```


DIKU
ds@di.ku.dk

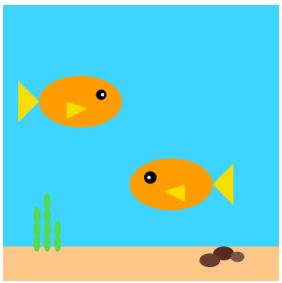
Change appearance

Try changing the `drawFish` function to strip the direction with the fish as an argument and use a condition to draw the fins and The eyes of the fish differ depending on the direction it is swimming.

```
function drawFish(x, y, eyeSize,
    velocity){
    ... draw body ...

    // Swim to the right
    if (velocity >= 0){
        ...draw fins and eyes ...
    }

    // Swim to the left
    if (velocity < 0){
        ...draw fins and eyes ...
    }
}
```



Finish the aquarium project

Now, we are almost done with the aquarium project. Add, if necessary, more elements. For example bubbles that emerge from the bottom and swim towards the surface, a chest with gold or whatever you want in your aquarium.

DIKU
ds@di.ku.dk

Animation example

You can use this code as a stepping stone to move forward

```
// ####### GLOBAL VARIABLES #######
var fish1X = 50;
var fish1Velocity = 1;
var fish2X = 500;
var fish2Velocity = -1;

// Canvas setup
function setup() {
  createCanvas(400, 400);
}

// ####### DRAWING FUNCTIONS #######

function drawFish(fishX, fishY, eyeSize, velocity) {
  // TODO: Complete this function to draw a fish
  // Remember to draw different fins/eyes based on velocity
      direction

  noStroke();
  fill(255, 157, 0);  // Orange body
  ellipse(fishX, fishY, 120, 75);

  if (velocity >= 0) {
    // Swimming right - draw fins and eye on right side
    // YOUR CODE HERE: Add tail, fins, and eye for right-facing fish

  } else {
    // Swimming left - draw fins and eye on left side
    // YOUR CODE HERE: Add tail, fins, and eye for left-facing fish

  }
}

function drawStones(stonesX) {
  // TODO: Draw a group of stones at the given x position
  // Hint: Use multiple ellipses with brown colors

  // YOUR CODE HERE
}
```

Animation example continued

```
function drawSeaweed(seaweedX) {
  // TODO: Draw seaweed plants at the given x position
  // Hint: Use green ellipses stacked vertically

  // YOUR CODE HERE
}

function drawScene() {
  // Draw the aquarium background
  background(61, 213, 255);  // Blue water

  // Draw sandy bottom
  fill(255, 199, 135);  // Sandy color
  noStroke();
  rect(0, 350, 400, 200);

  // Add decorations
  drawStones(300);
  drawSeaweed(65);
}

// ###### END DRAWING FUNCTIONS ######

// ####### CONTROL FUNCTIONS #######

function moveFish1() {
  // Update fish1 position
  fish1X = fish1X + fish1Velocity;

  // Check boundaries and reverse direction
  if (fish1X > 500) {
    fish1Velocity = -fish1Velocity;
  }
  if (fish1X < 0) {
    fish1Velocity = -fish1Velocity;
  }
}

function moveFish2() {
  // TODO: Complete this function for fish2
  // Should work similar to moveFish1 but use fish2 variables

  // YOUR CODE HERE
}

// ######## END CONTROL FUNCTIONS #######
```


DIKU
ds@di.ku.dk

Animation example continued

```
// ########## MAIN PROGRAM ##########

function draw() {
  // Update positions
  moveFish1();
  moveFish2();

  // Draw everything
  drawScene();
  drawFish(fish1X, 140, 15, fish1Velocity);
  drawFish(fish2X, 260, 30, fish2Velocity);
}
```

More exercises about conditions

Open the Green City project and do the following:

- Make the cloud move back to the start and pass by again and again, each time it hits the edge
- Make the car turn over at both ends.

Tasks

The car must stop when the battery is empty

- Define a global variable `carBattery` and set it to 100
- Decrease it by 0.1 each time `draw` is called
- Display the battery status using the `text(string, x, y)` function. Remember to convert the number to a string using `str()`.
- If the battery is empty (`<= 0`) the car must stop (set velocity to 0)

Add a `keyPressed()` function and make it possible to charge the electric car when you press 'C':

- Add `0.3` to `carBattery` every time 'C' is pressed on the keyboard
- If `carBattery` exceeds 100, set it to 100 so you can't charge more than 100%

DIKU
ds@di.ku.dk

Changing wind speed

The variable `frameCount` counts how many times `draw` is run since the program started. Enter this in the draw function i the electric car project:

```
fill(0, 0, 0);
text(frameCount, 350, 20);

if (frameCount % 60 == 0){
    print(frameCount)
}
```

Since `draw` is executed at *60 frames per second*, the **`print`** function will be executed every second.

We can also use that in the electric car project to update the wind speed every second.

- Create a global windSpeed variable
- Update windSpeed with a new random value every second
- Display the wind speed using `text()`
- Create a global variable `powerplantOn`
- Make it possible to turn the power plant on and off with a press of 'p'
- Draw clouds of smoke over the power plant when it is on
- Create a variable `co2emission` and add a bit as long the power plant is on. Display the value with `text()`

DIKU
ds@di.ku.dk

Sounds

In this small worksheet, we will add sound files to make sound effects!

1. First check reference https://p5js.org/reference/p5.MediaElement/play/
2. Then download the sound files from ABASALON
3. Also check out the sound sketch also uploaded
4. Create a new file named `sound01`

```
function preload() {
    // Load the sound files before the program starts
    // Load the first sound file (e.g., a music track or sound
       effect)
    sound = loadSound('sound.mp3');
    // Load the second sound file (e.g., a ball tap sound)
    sound1 = loadSound('balltap.wav');
}

function setup() {
  // Setup the canvas size
  createCanvas(400, 400);  // Create a canvas of 400x400 pixels
}

function draw() {
  // Draw the background and instructions
  background(220);  // Set the background color to light grey (RGB
     value: 220)
  // Align text to the center horizontally and vertically
  textAlign(CENTER, CENTER);
  textSize(24);  // Set the text size to 24 pixels
  // Display the instructions in the center of the canvas
  text('Press "S" or "A" to play sound', width / 2, height / 2);
}

function keyPressed() {
  // Detect key presses and play the corresponding sound
  if (key === 's' || key === 'S') {
    // Play the first sound when the 'S' key is pressed (case
       insensitive)
    sound.play();
  }
  if (key === 'a' || key === 'A') {
    // Play the second sound when the 'A' key is pressed (case
       insensitive)
    sound1.play();
  }
}
```

Descriptions

- **'preload()' function**: Loads the sound files into memory before the sketch starts. This ensures that the sounds are ready to be played when triggered.
- **'setup()' function**: Sets up the canvas where the sketch will be displayed, specifying its size (400x400 pixels).
- **'draw()' function**: Continuously runs in a loop, updating the background color and displaying the instruction text centered on the canvas.

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.

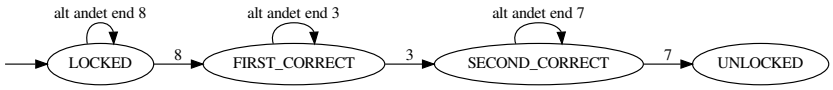
Finite state machines - Task

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.

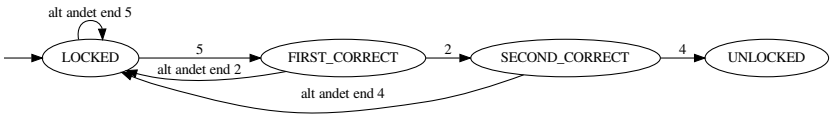
Finite state machines - Padlock

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:

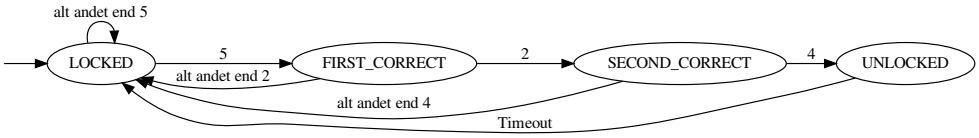


- Draw an extended state diagram of the combination lock with an extra mode, so 4 digits are required. Next, add the extra state in the code.

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