Hello Wednesday

Kickstart-kursus i programmering dag 3

Daniel Spikol

ds@di.ku.dk

DIKU \ Københavns Universitet

August 2025

Recap from Tuesday

- Python and Functions
- Animations
- Pair Programming
- Mouse and Key Input

Wednesday IFOs

- Variable Scoping
- Draw() function
- Conditionals and FSM
- Sounds
- Ideation

Brainstorming Schedule

- 12:30 Team Charlie
- 12:55 Team Beta
- 13.20 Team Alpha

Variable Scoping Processing

- The principles of variable scoping in Processing.py largely follow those of JavaScript but with some considerations given the environment of Processing's draw loop and function setup.
- Scoping in programming refers to the region or portion of the code where a variable or function is defined and can be accessed or modified.
- The concept of scoping is crucial for understanding variable lifetimes, visibility, and potential naming conflicts in your programs. It helps manage and organize data and functionality, enabling modular and maintainable code designs.

Variable Scoping Processing

Global Variables

Global Variables: Variables declared outside of any function are global to the sketch. They can be accessed and modified from any function, but if you want to modify them inside a function, you must declare them as `global` within that function.

Global Variable example

```
let globalVarX = 50;

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

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

Variable Scoping Processing

Local Variables

Local Variables: Variables declared inside a function are local to that function. They cannot be accessed outside of the function, and their memory is reclaimed once the function execution is complete.

```
function setup() {
  createCanvas(400, 400);
  let myVar = 42;
  print(myVar); // This prints 42
}

function draw() {
  background(220);
  print(myVar); // This returns error, not defined
}
```

Debugging Strategies

Debugging in Programming

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 - 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()
```

Debugging Tools & Techniques

Use console.log() to Track Variables

```
let circleX = width / 2;
console.log('circleX: ' + circleX);  // Check the value!

function draw() {
  console.log('frameCount: ' + frameCount);  // Track animation
  circle(circleX, 100, 50);
}
```

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)

Best Debugging Practices

Start Small and Test Often

- **Don't write your whole program at once!**
- Test after every few lines of code
- It's easier to find bugs in code you just wrote
- Create small example programs (MCVE*) to isolate problems
  (**Minimal Complete Verifiable Example*)

Other Powerful Techniques

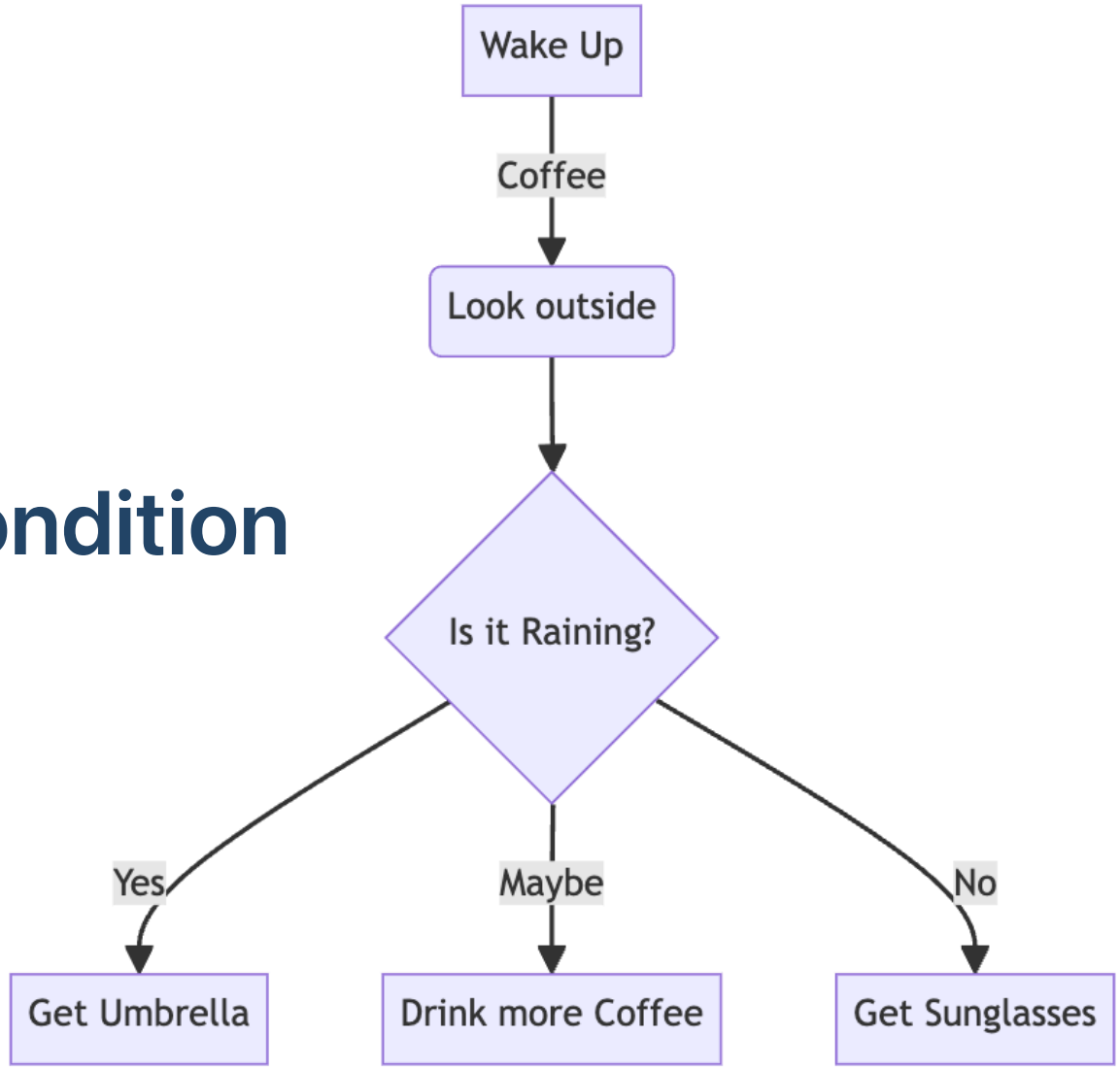
- **Rubber Duck Debugging** 🦆 - Explain your problem out loud to anything
- **Take Breaks** - Solutions often come when you step away
- **Use Browser Developer Tools** - Press F12 for advanced debugging
- **Comment Out Code** - Use `//` to temporarily disable problematic sections

Remember: Debugging is a Normal Part of Programming!

"Making mistakes and fixing them is how we learn and grow as programmers"

Conditionals

Flowchart Condition



Conditionals in Computer Science

- Conditionals in computer science refer to constructs that allow for decision-making in code.
- They determine the flow of execution based on whether a given condition is true or false.
- Depending on the outcome of the condition, different blocks of code will be executed.
- Specifically, conditionals perform different computations or actions depending on whether a programmer-defined boolean condition evaluates to true or false.

Conditionals Conceptually

- **Basic Conditional (IF)**
  - **Concept**: If a specific condition is true, do something.
  - **Example**: "If it's raining, take an umbrella."
- **Alternative Path (ELSE)**
  - **Concept**: If the first condition isn't met, do something else instead.
  - **Example**: "If it's raining, take an umbrella. Otherwise, wear sunglasses."
- **Multiple Conditions (ELSE IF or ELIF)**
  - **Concept**: Check multiple conditions in sequence, and do the first thing that's true.
  - **Example**: "If it's raining, take an umbrella. If it's sunny, wear sunglasses. Otherwise, just go outside as usual."

Conditionals

- **If Statement:** Executes a code block if a specified condition is true.
- **Else Statement:** Used in conjunction with an if statement, it specifies a block of code to be executed if the condition in the if statement is false.
- **Else If Statement:** Used to specify a new condition to test if the first condition is false.
- **Switch or Case Statement:** Allows a variable to be tested for equality against a list of values.

Conditionals in p5.js

p5js supports the usual logical conditions from mathematics:

- Equals: `a == b` (Loose Equality Operator)
- Equals: `a === b` (Strict Equality Operator)
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`
- `||` checks multiple conditions and returns true if any one of the conditions is true.

Conditionals in p5.js I

if:

```
if (x > 10){
    console.log("x is greater than 10")
}
```

else:

```
if (x > 10) {
    console.log("x is greater than 10");
  } else {
    console.log("x is 10 or less");
  }
```

Conditionals in p5.js II

else if:

```
let x = 13;
  if (x > 10) {
    console.log("x is greater than 10");
  } else if (x == 10) {
    console.log("x is 10 or less");
  }
```

You can also combine conditions using logical operators (`and, or, not`):

```
if (x > 10 && y<5){
    console.log("x is greater than 10");
  }
```

Conditionals in p5.js

For instance, to animate a circle moving across the screen and to make it wrap around when it reaches the edge:

```
let x_pos = 0; //set the x postion

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

function draw() {
  background(220);
  ellipse(x_pos, height / 2, 50, 50);
  x_pos += 2;
  if (x_pos > 400) {
    x_pos = 0; // resets x position
  }
}
```

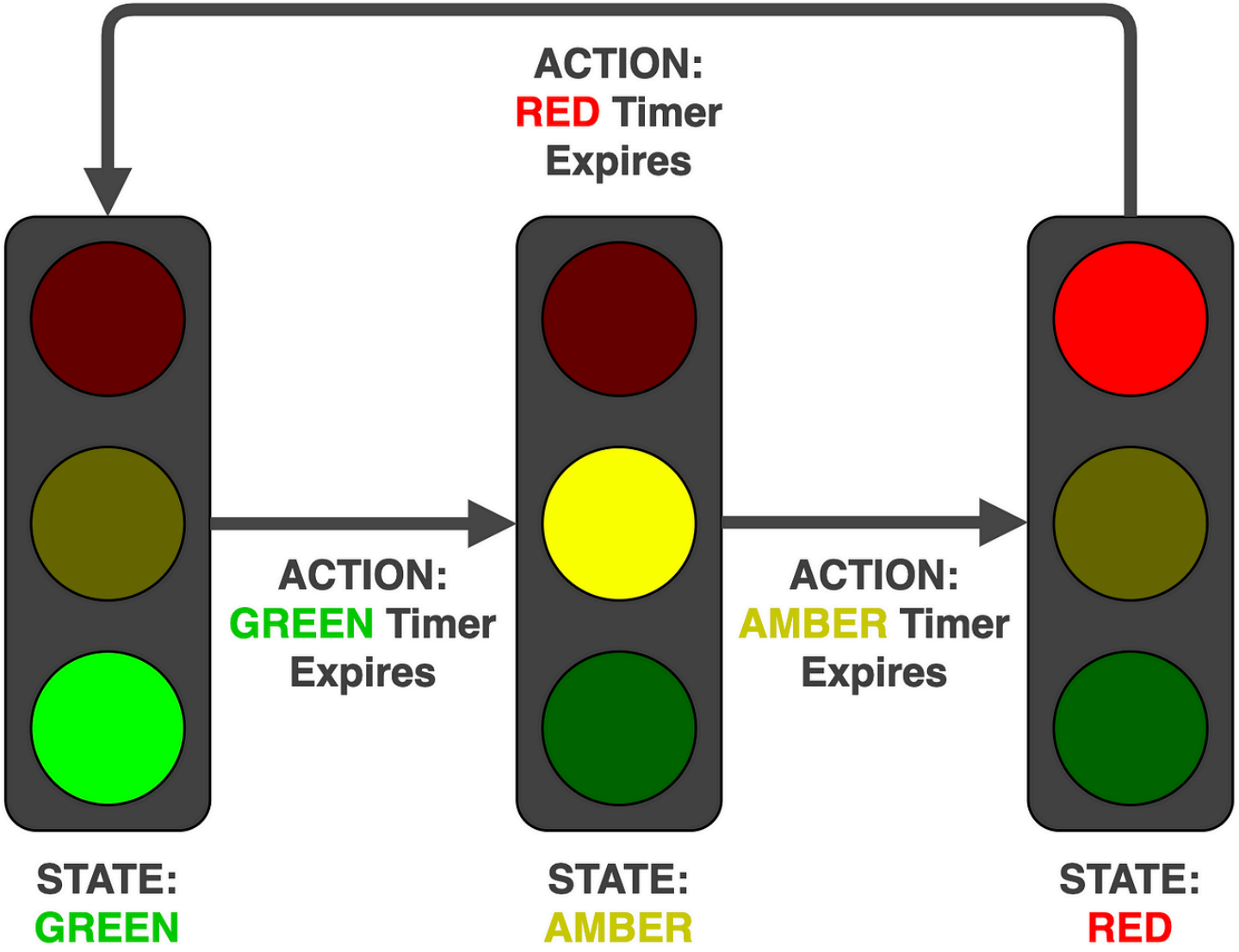
Finite State Machines- FSM

- A Finite State Machine (FSM) is a mathematical model of computation used to design algorithms.
- In the context of computer games, FSMs are often used for character behaviour, where different states might represent actions like "idle", "attack", "defend", or "flee", and game events or conditions determine transitions between states.

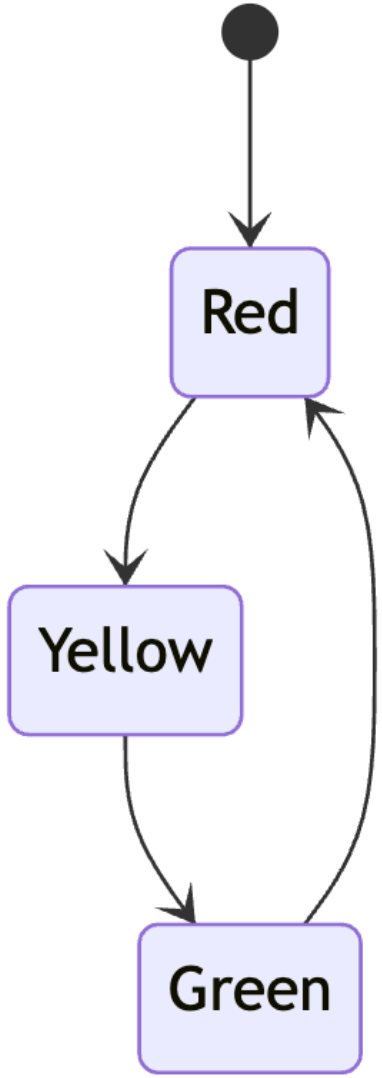
The STATES in a FSM

- **Discrete States:** An FSM consists of a limited or finite number of states. It can be in just one of these states at any given moment. Transitions define how it changes from one state to another based on inputs or conditions.
- **Transitions & Triggers:** Events or conditions trigger transitions between states. Each state specifies which state the machine will move to next for each possible input.
- **Start and End States:** Among the finite states, there is one initial state where the FSM begins its operation. Additionally, there can be one or more end states where the FSM is considered to be completed or final.

The Classic Example



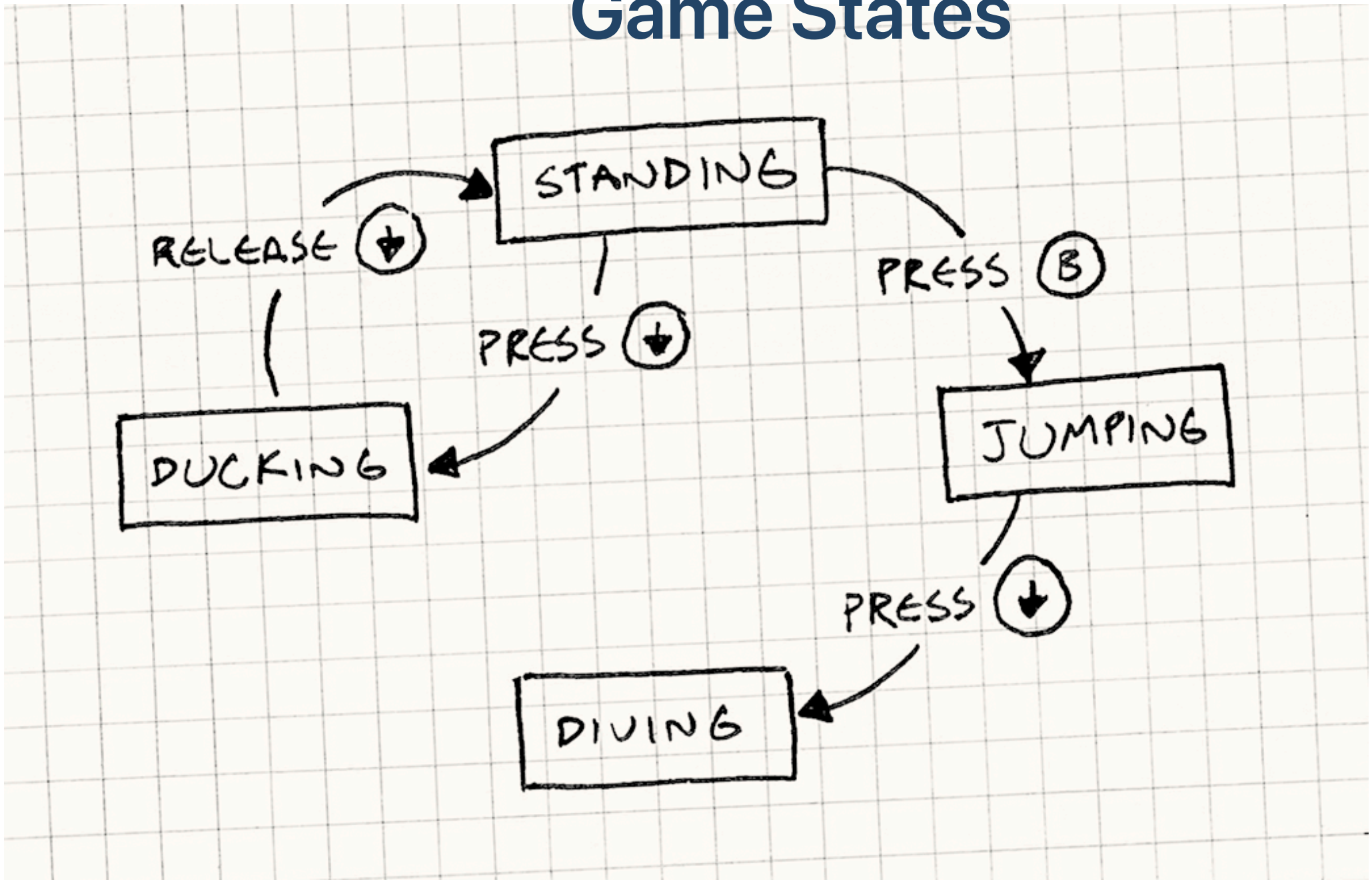
State Diagram



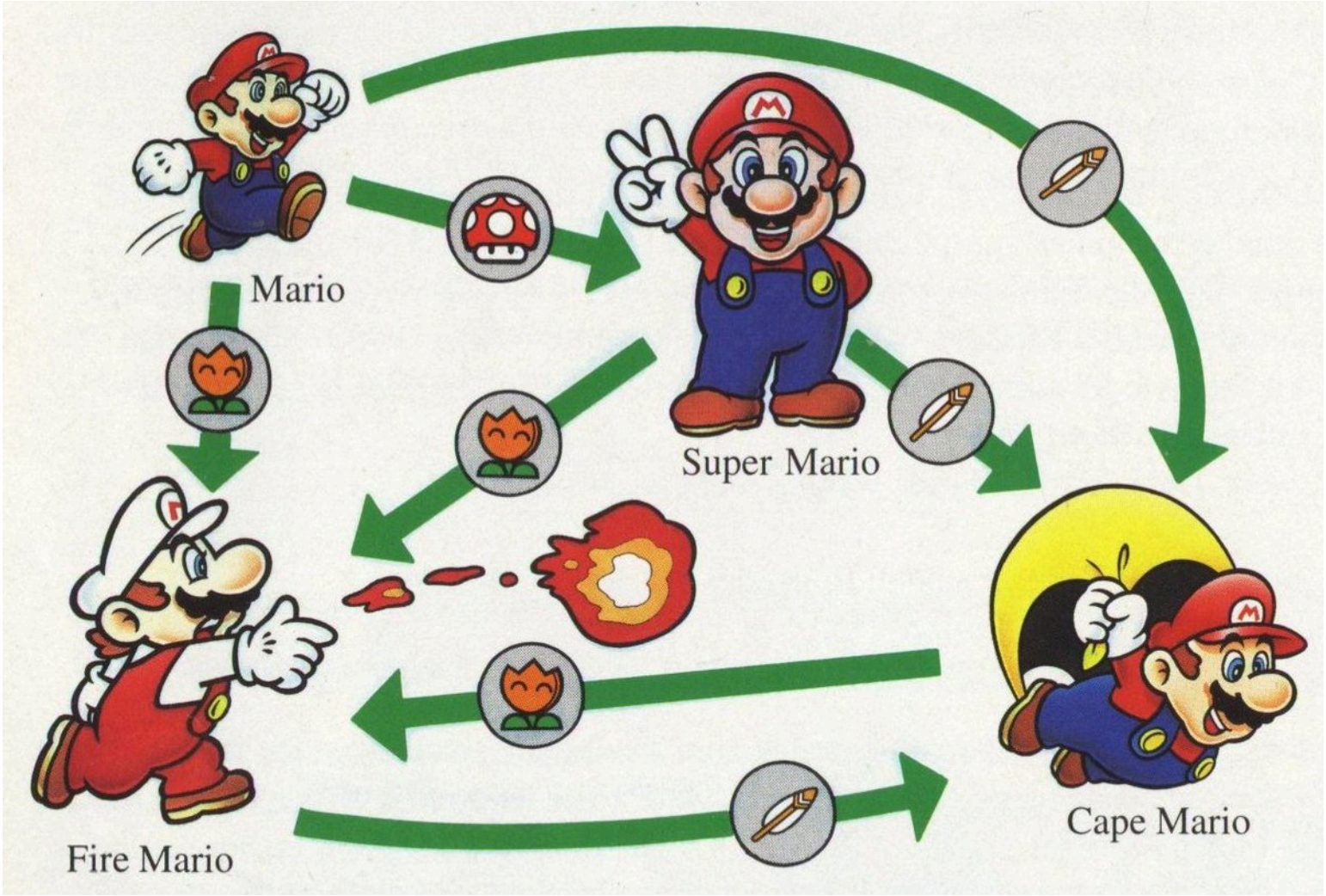
Mathematical Abstraction of the FSM

- A finite state machine is a mathematical abstraction used to design algorithms. In simple terms, a state machine will read a series of inputs.
- When it reads an input, it will switch to a different state. Each state specifies which state to switch to for a given input.-

Game States


STANDING
RELEASE
PRESS
PRESS
B
DUCKING
JUMPING
PRESS
DIVING

Mario States



FSM Code Example

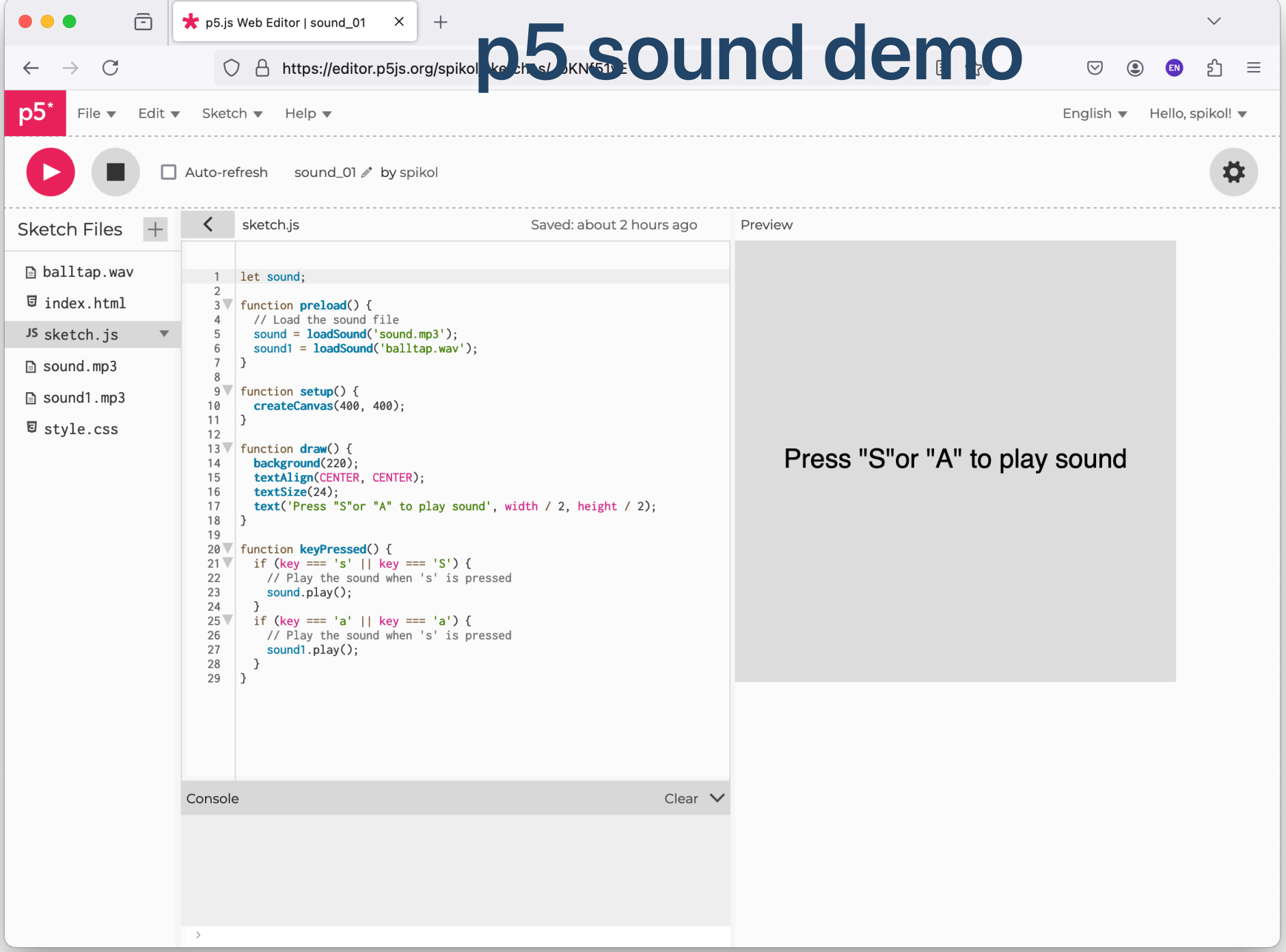
```
//global vars
let state_on = "ON";
let state_off = "OFF";

//initial state
current_state = state_off;

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

function draw() {
  background(220);

  if (current_state == state_on) {
    fill(0, 255, 0); // green is for on
  } else if (current_state == state_off) {
    fill(255, 0, 0); // red is for off
  }
  ellipse(width / 2, height / 2, 100, 100);
}

function mousePressed() {
  if (current_state == state_off) {
    current_state = state_on;
  } else if (current_state == state_on) {
    current_state = state_off;
  }
}
```

How do you add Yellow?

Sound in p5.js

- Let's keep it simple, but if interested explore the p5 reference
- We are going to load and play sounds, plenty other ways to work with sound.
- SoundFile object with a path to a file.
- The p5.SoundFile may not be available immediately because it loads the file information asynchronously.
- To do something with the sound as soon as it loads pass the name of a function as the second parameter.
- Only one file path is required. However, audio file formats (i.e. mp3, ogg, wav and m4a/aac) are not supported by all web browsers.

p5 sound demo



Code

```
function preload() {
  // Load the sound file
  sound = loadSound('sound.mp3');
  sound1 = loadSound('balltap.wav');
}

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

function draw() {
  background(220);
  textAlign(CENTER, CENTER);
  textSize(24);
  text('Press "S"or "A" to play sound', width / 2, height / 2);
}

function keyPressed() {
  if (key === 's' || key === 'S') {
    // Play the sound when 's' is pressed
    sound.play();
  }
  if (key === 'a' || key === 'a') {
    // Play the sound when 's' is pressed
    sound1.play();
  }
}
```