Let's say we make a 2-player game. When the number of registered players reaches 2, we want the game state to become 1 (PLAY). When the game state changes to 1, we would want the game to start.

There are a number of ways in which we can go about doing this. We can start writing the code immediately. But good programmers, before writing code, think about how to structure their code.

programming style are we using in our codes so far is **OOPs** - **object**-**oriented** style

pages which ask for users to input information are called Forms.

We need to have at least 3 objects:

1. **Form**: The form should contain the input box and a button to log in.

● When the button is pressed, the player's name should be registered in the database and a new player should be created.

2. **Player**: A new player object should be created every time a new player logs in. It should contain all the information about the player - name, position in the game, and so on. We can add multiple properties about the player as required by the game.

3. **Game Object**: The game object should be able to hold the state of the game. It should be able to display form when the game state is 0(WAIT), the game when the game state is 1(PLAY), and the leaderboard when the game state is 2(END). To start, we will only consider the case when the game state is 0.

One new function here; The **windowResized**() function in p5.js is called once every time the browser window is resized. It adjusts its height and width automatically whenever the size of the window is increased. This function is invoked automatically as soon as the window is resized and then creates a new canvas corresponding to it.

As you can see the game has a js folder which contains all three object files which we discussed earlier. These files are added to index.html

● form.js

● game.js

● player.js

**Form Class**.

**HTML** is used to create any content like a form on a page. **HTML** is similar to markdown in some ways. An **HTML** contains elements that define the structure of a page. A simple html page contains: - head: where all the scripts and stylesheets for the page are added. - body: where all the content of the page is added. The body of an HTML page can contain several different types of elements: - **h1, h2, h3**: display **headings** of different sizes. - input: to collect input from the user. - button: to display a button. This model of an HTML page is called the **Document Object Model (or DOM).** We will be using the p5 Dom library to create the form

Each element is created in the **constructor**() in the Form class. **this.input**, will display an empty box with the text given in the placeholder attribute.

.**hide**() is used with each element to hide/ remove the element from the canvas.

How to take action when the button is pressed by the mouse.

Players will press the button using the mouse; in p5 there is a **mousePressed**() function which is called whenever the mouse is pressed.

When the button is pressed, we want to **greet** the player when the player writes their name and logs in. We also want to **update** the **playerCount** and the **player nam**e in the **database**, this part we will cover in the next class. button.mousePressed() can be used to trigger an action when a mouse is pressed on the button. It expects a function as an argument. Let's write the code to display a greeting and update the database when the button is pressed. We will create a new function **handleMousePressed**(){ } Remember to call this function in display().

**Arrow function.**

Arrow functions allow us to write shorter function syntax:

Before:

hello = function() {  
  return "Hello World!";  
}

### With Arrow Function:

hello = () => {  
  return "Hello World!";  
}

It gets shorter! If the function has only one statement, and the statement returns a value, you can remove the brackets and the return keyword:

### Arrow Functions Return Value by Default:

hello = () => "Hello World!";

If you have parameters, you pass them inside the parentheses:

### Arrow Function With Parameters:

hello = (val) => "Hello " + val;