ADV-C137

REALTIME OBJECT IDENTIFICATION

What is our GOAL for this MODULE?

We learned how to import a video in p5.js canvas, and how to do object detection on the imporated video, and create a web app out of it.

What did we ACHIEVE in the class TODAY?

- Read the objects array and draw rectangles and place the labels.

Which CONCEPTS/ CODING did we cover today?

- Added code for gotResult() function.
- Added code of for-loop for reading the objects and drawing rectangles and placing the labels.

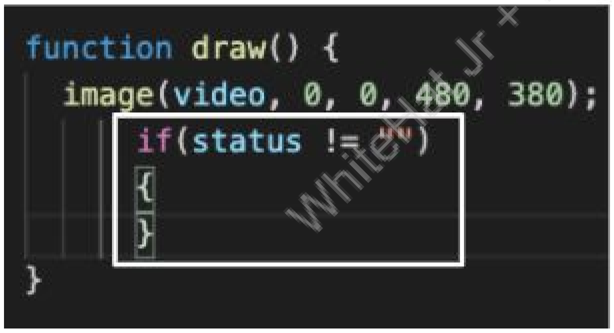
How did we DO the activities?

1. First add an "**if condition**" for checking if the cocossd model is loaded or not.So the purpose of this "**if condition**" will be that the process of drawing should start after the model is loaded.

Status variable, as it is used to keep a track that model is loaded or not, also we are setting it's value as true after the **cocossd** model is loaded.
As we are setting the value of the **status** variable as **true** after the **cocossd** model is loaded.
So this means that the **status** variable is empty at the very beginning of the code, and when the **cocossd** model is loaded the value of the **status** variable becomes **true.**
Using this knowledge we can write a "**if condition**" to check if the **status** variable is not empty(which means the **status** variable is **true** -- which means the model is loaded because after the model is loaded the **status** variable becomes **true**) then start drawing.
So "**if condition**" will be - "**if**" the **status** variable is not equal to empty then start drawing the rectangles.

Write this "**if condition**" inside p5.js **draw()** function, the reason of this is we want the object detection to be a continuous process, and **draw()** function is called continually, and inside this "**if condition**" we will be writing the code of detecting objects, that's why we have written this "**if condition**" inside draw() function.

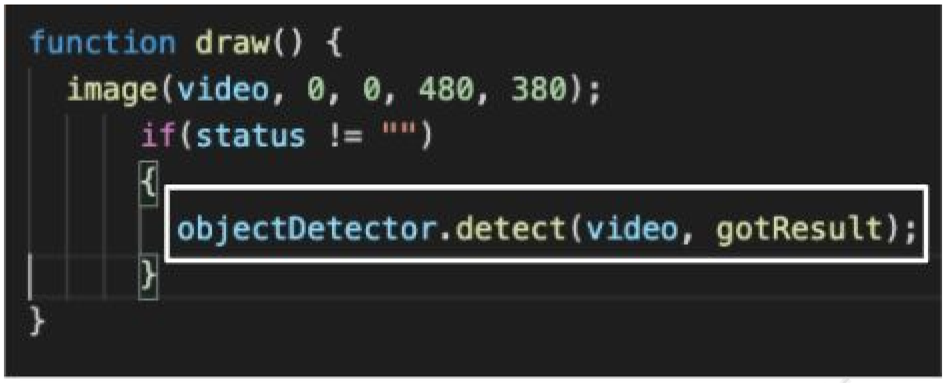
Code:



```
function draw() {
  image(video, 0, 0, 480, 380);
    if(status != "")
    {
    }
}
```

So in the "**if condition**" we are checking that "if"[`if(`] **status** variable[`status`] is not[`!=`] equal [`=`] to empty[`"")`] then it should go inside this "if condition" and start drawing rectangles.

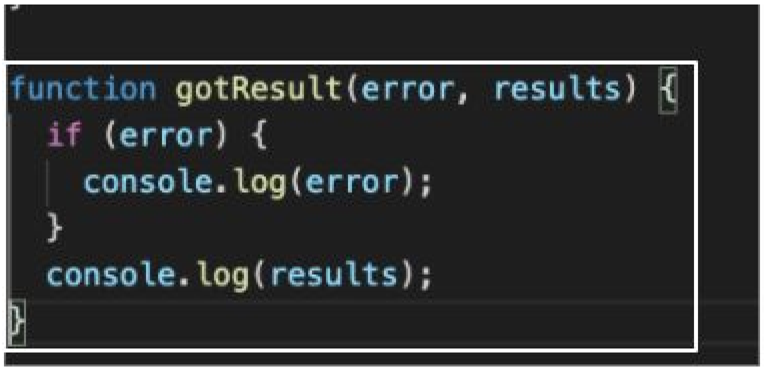
2. Now inside this "**if condition**" write code for executing cocossd mode.



```
function draw() {
  image(video, 0, 0, 480, 380);
    if(status != "")
    {
      objectDetector.detect(video, gotResult);
    }
}
```

- `objectDetector` is the variable that is holding the **cocossd** model which we had defined in the previous class.
- `detect` - it is a predefined function of ml5.js used for object detection, and gets back the result of object detection. In `detect` function we are required to pass two parameters:
  - The video which we had loaded in the previous class.
    `objectDetector.detect(video,` The video variable holds the video which we had loaded in the previous class.
  - A function, which will hold the result of object detection
    `detect(img, gotResult);` **gotResult** function will be holding the result of object detection, this function define in the next step.

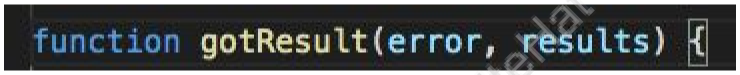
3. Now write the code of the **gotResult()** function.The purpose of this function is to show the result which has been achieved after the object detection. Add **gotResult().**



```
function gotResult(error, results) {
  if (error) {
    console.log(error);
  }
  console.log(results);
}
```

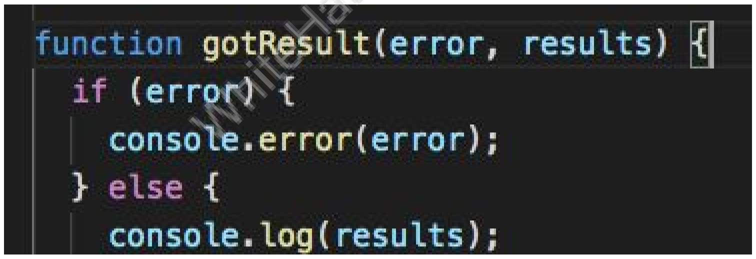
- This **gotResult** function which holds the result of the object detection, has two parameters inside it one is error and second is results.

- So when we define the **gotResult** function we need to pass error and results inside the function.



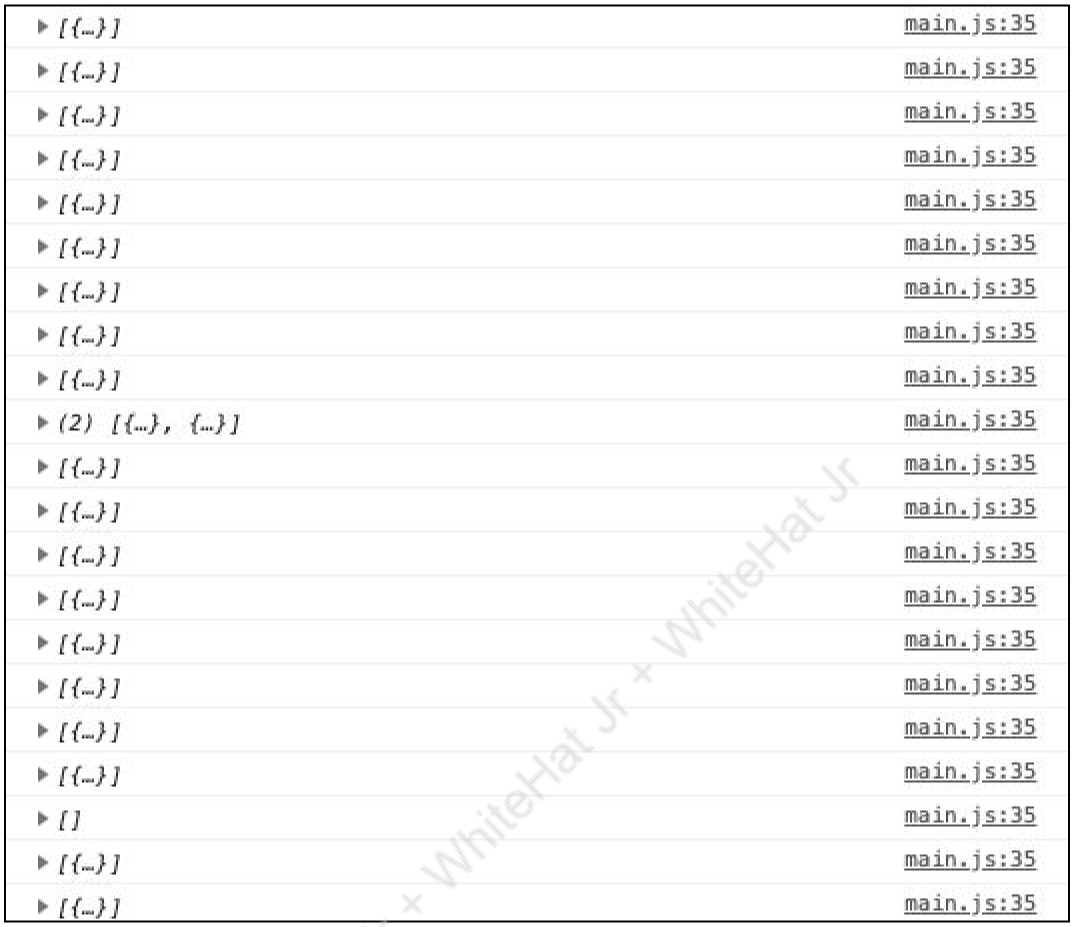
```
function gotResult(error, results) {
```

- Now check if there is an error. If yes, then console the error, else console the results, and see what we get in the result.



```
function gotResult(error, results) {
  if (error) {
    console.error(error);
  } else {
    console.log(results);
```

- Output on console screen:

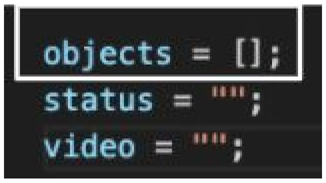


```
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
▸ (2) [{…}, {…}]                          main.js:35
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
▸ []                                      main.js:35
▸ [{…}]                                   main.js:35
▸ [{…}]                                   main.js:35
```

4. Create an empty array, and assign the **results** array obtained from object detection to this empty array.

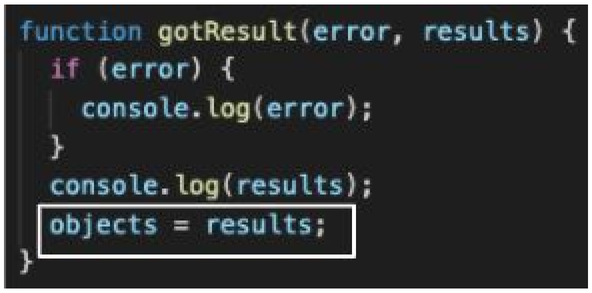
As you know we have to draw rectangles around all the objects and also label them. And for drawing rectangles we have to use the data which we get from the **results** array.
And the **results** array is inside **gotReults()** function.
But we are supposed to add code for drawing rectangles inside the **draw()** function.
Because all the code for drawing anything on canvas should be written inside the **draw()** function.
Which means somehow we have to access the values of the **results** array inside the **draw()** function so that we can draw rectangles around all the objects using the values of the **results** array.
For achieving the above task assign the **results** array obtained from object detection to this empty array, and **then use this array which we have defined,** inside the **draw()** function for drawing rectangles around the object.

5. Defining empty array at the beginning of **main.js** file.



```
objects = [];
status = "";
video = "";
```

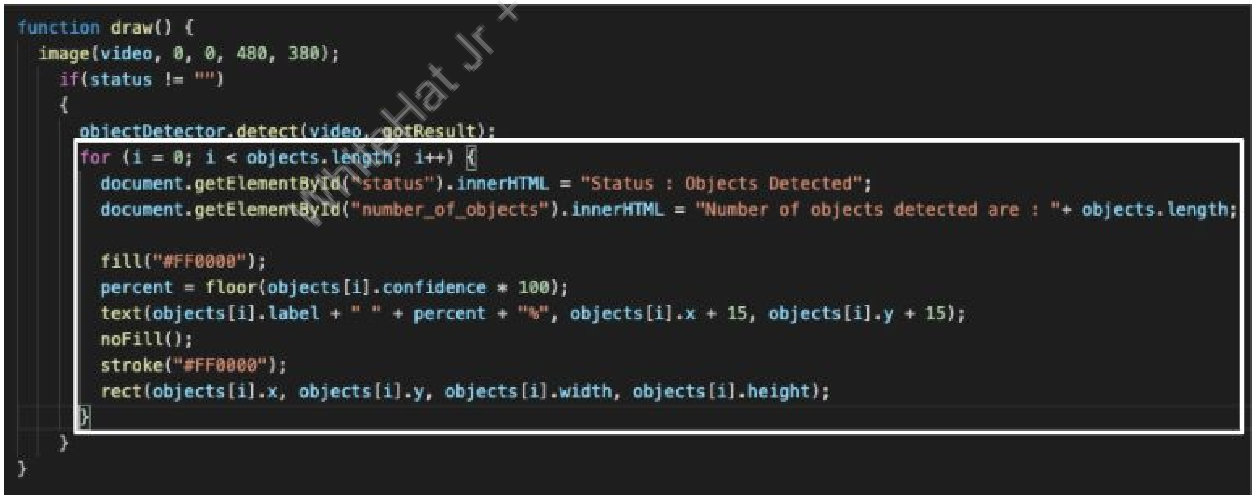
6. Assigning the **results** array to **objects** array.



```
function gotResult(error, results) {
  if (error) {
    console.log(error);
  }
  console.log(results);
  objects = results;
}
```

- After doing this we can use the **objects** array inside the **draw()** function, and using the values of the **objects** array we can draw rectangles around all the objects and also label them.

7. Now we can start adding code for drawing rectangles around the detected object and also add labels to those objects.

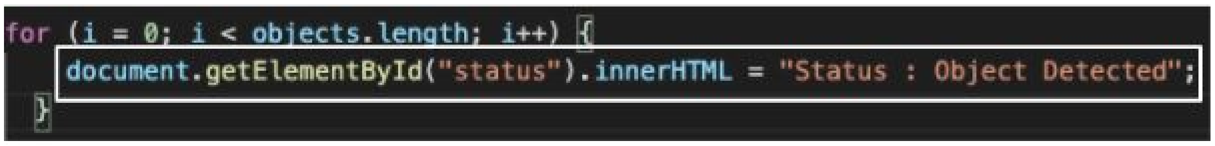


```
function draw() {
  image(video, 0, 0, 480, 380);
    if(status != "")
    {
      objectDetector.detect(video, gotResult);
      for (i = 0; i < objects.length; i++) {
        document.getElementById("status").innerHTML = "Status : Objects Detected";
        document.getElementById("number_of_objects").innerHTML = "Number of objects detected are : "+ objects.length;

        fill("#FF0000");
        percent = floor(objects[i].confidence * 100);
        text(objects[i].label + " " + percent + "%", objects[i].x + 15, objects[i].y + 15);
        noFill();
        stroke("#FF0000");
        rect(objects[i].x, objects[i].y, objects[i].width, objects[i].height);
      }
    }
}
```

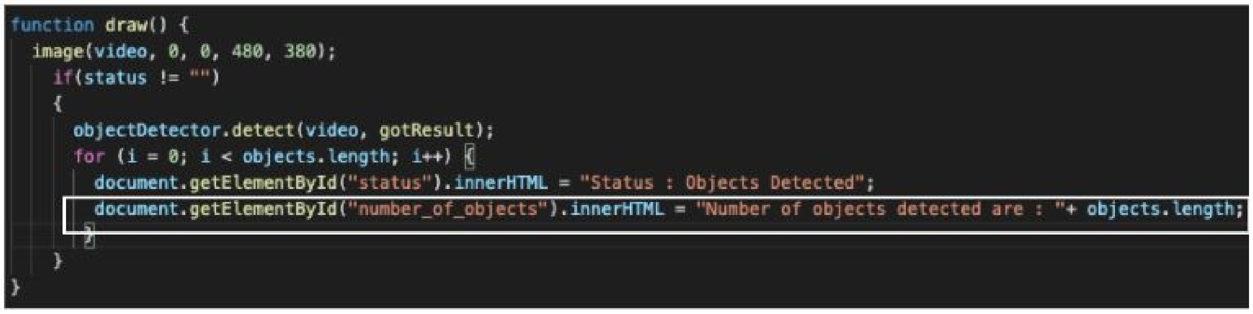
Use a for-loop to read the objects array for fetching the values like - label, confidence, x and y coordinates of the objects, and using these values draw a rectangle around the detected object and label them.

8. Defining for loop - `for (i = 0; i < objects.length; i++)`.
    - For loop should start at 0 - `i = 0` - because the array index also starts with zero.
    - Pass `i < objects.length` in the second parameter of the for-loop. The purpose of using a for-loop is that we should be able to fetch all the objects from the **objects** array that's why we can't give a number in the second parameter(which holds the value at which point the loop should stop). As we don't know how many objects there are in a video.
        - If we fixed a given number as 3 and there are more than 3 objects in the image then, the purpose of the for-loop **fails.**
        - If we fixed a given number as 3 and there are less than 3 objects in the image then, the purpose of the for-loop **fails.**
    - That's why we have written `i < objects.length` - meaning the for-loop will stop at the length of the **objects** array.
        - If the length of the **objects** array is 3 which means - **i < 3**
        - If the length of the **objects** array is 5 which means - **i < 5**
        - If the length of the **objects** array is 2 which means - **i < 2**
    - In the third parameter we have to write how much interval should be there between each loop - we have mentioned `i++`. These two plus (**++**) signs means increment by 1. So the interval should be 1.
9. If you remember in the class number - 135 we had defined an h3 tag and gave its ID as a **status**, and the purpose of this h3 tag was to show the status.
   Now the objects are detected and in the next step start writing code for drawing rectangles around the objects and also give labels and confidence, but before we need to update this h3 tag with "**Status : Objects Detected**" this will indicate that the objects are detected.



```
for (i = 0; i < objects.length; i++) {
    document.getElementById("status").innerHTML = "Status : Object Detected";
}
```

10. If you remember in the class number - 135 we had defined an h3 tag and gave its ID as a **number_of_objects** The purpose of h3 tag was to hold the numbers of objects detected, so let's update the **h3 tag** with numbers of objects detected.



```
function draw() {
  image(video, 0, 0, 480, 380);
    if(status != "")
    {
      objectDetector.detect(video, gotResult);
      for (i = 0; i < objects.length; i++) {
        document.getElementById("status").innerHTML = "Status : Objects Detected";
        document.getElementById("number_of_objects").innerHTML = "Number of objects detected are : "+ objects.length;
      }
    }
}
```

- First refer to that HTML element using its id -

  ```
  document.getElementById("number_of_objects").innerHTML =
  ```

- Then write the required text in double quotes

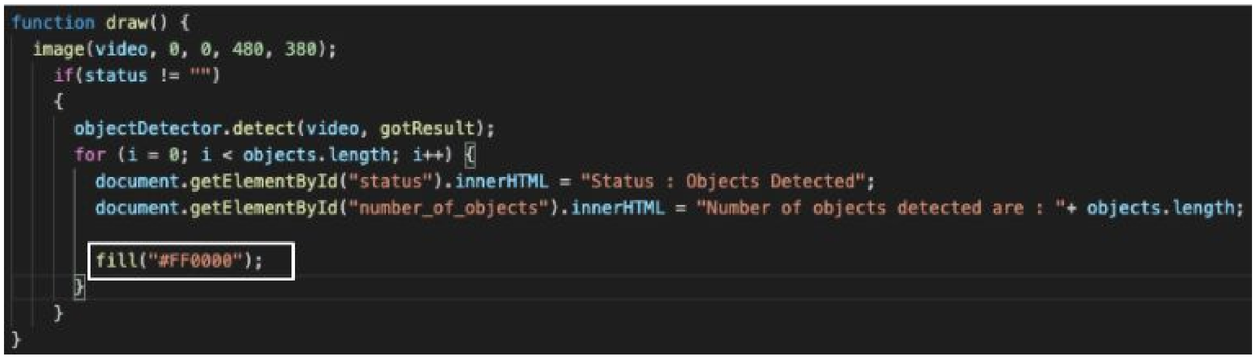
  ```
  "Number of objects detected are : "
  ```

- `objects.length` - it will have the length of the **objects** array. Length of **objects** means how many objects are there inside the **objects** array. And this number of objects inside **objects** array denotes how many images are detected. That is why we set the end condition of for-loop to **objects.length** as it tells how many objects are detected.

- So as a conclusion we can say that `objects.length` will give the number of objects detected, and henceforth we can use this to show on the h3 tag about how many objects are detected.

```
"Number of objects detected are : "+ objects.length;
```

11. Now we need to set the color for the text, and for that use **fill()** function, and inside it pass a HEX code of red color. If you want you can put any color.



```
function draw() {
  image(video, 0, 0, 480, 380);
    if(status != "")
    {
      objectDetector.detect(video, gotResult);
      for (i = 0; i < objects.length; i++) {
        document.getElementById("status").innerHTML = "Status : Objects Detected";
        document.getElementById("number_of_objects").innerHTML = "Number of objects detected are : "+ objects.length;

        fill("#FF0000");
      }
    }
}
```

12. Now we need to fetch the confidence and convert it into percentage.

Code for reading confidence from objects array - `objects[i].confidence`. For eg - Let's consider that there are 2 arrays inside the **objects** array. Meaning that the **length** of this array is 2.

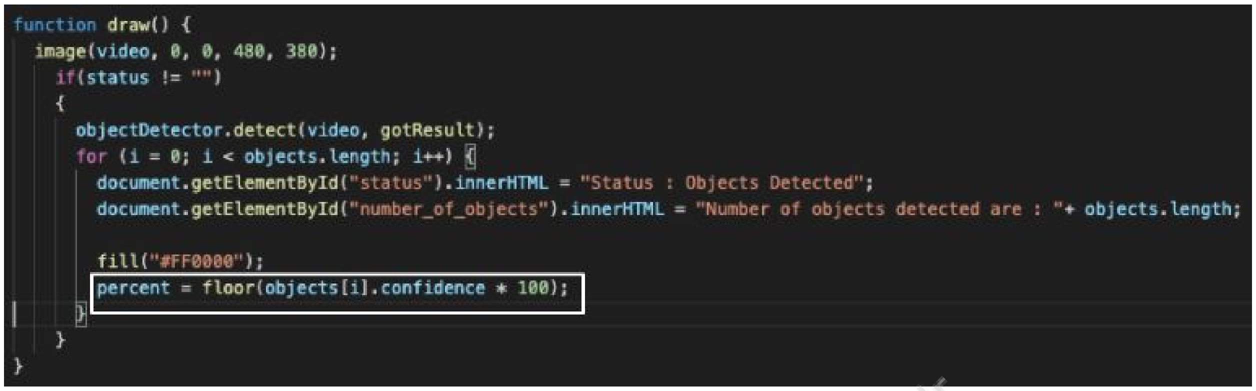
- **When the loop starts i = 0**
  - **objects[i].confidence** will become objects[0].confidence // this means we got the confidence of the first object.
- **Then i is incremented and i = 1**
  - **objects[i].confidence** will become objects[1].confidence // this means we got the confidence of the second object.
- **Then i is incremented and i =2**
  - The **length** of the array is 2, means the loop terminates.

From the above example we can consider that **objects[i].confidence** will fetch the confidence of all the objects one by one from the **objects** array.
Now we need to convert this into percentage, so for that multiply **objects[i].confidence** by 100, so that we get a value in percentage `object[0].confidence * 100`.

As the confidence has a lot of decimals `confidence: 0.8548185229301453`
After we converted into percentage still it will have lot of decimal points
So to remove these decimals use p5.js **floor()** function.
And if we don't remove these decimals then a lot of decimals will come on canvas screens, and we don't want that.

floor() function will remove all the decimals `floor(objects[i].confidence * 100);`

13. Now store this value inside a variable so that we can use this confidence value in the next step.



```
function draw() {
  image(video, 0, 0, 480, 380);
    if(status != "")
    {
      objectDetector.detect(video, gotResult);
      for (i = 0; i < objects.length; i++) {
        document.getElementById("status").innerHTML = "Status : Objects Detected";
        document.getElementById("number_of_objects").innerHTML = "Number of objects detected are : "+ objects.length;

        fill("#FF0000");
        percent = floor(objects[i].confidence * 100);
      }
    }
}
```

14. Next step is to fetch the labels from the array and display near the object in the video. For displaying the text near the object in the image, we have to use the **text()** function. Syntax of **text()** function:
text("text what we want to show on the canvas", x coordinate, y coordinate)

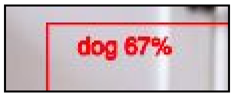
Fetching label and displaying label, confidence, and percentage symbol.

Code for reading confidence from objects array - `objects[i].label`. For eg - Let's consider there are 2 arrays inside the **objects** array. Means the **length** of this array is 2.

- **When the loop starts i = 0**
  - **objects[i].label** will become objects[0].label // this means we got the label of the first object.
- **Then i is incremented and i = 1**
  - **objects[i].label** will become objects[1].label // this means we got the label of the second object.

- **Then i is incremented and i =2**
  - The **length** of the array is 2, means the loop terminates.

As you see on canvas we display the label, confidence, and the percentage symbol like this:



- The label of the object - the code will be `text(objects[i].label`.

- A space - the code will be - `text(objects[i].label + " "`
- Then percentage - the code will be - `text(objects[i].label + " " + percent`.
- The the symbol of the percentage - the code will be -

```
text(objects[i].label + " " + percent + "%",
```

Fetching x coordinates and passing inside text() function

Code for reading confidence from objects array - `objects[i].x` because for eg - Let's consider that there are 2 arrays inside the **objects** array. Means the **length** of this array is 2.

- **When the loop starts i = 0**
  - **objects[i].x** will become objects[0].x // this means we got the x coordinate of the first object.
- **Then i is incremented and i = 1**
  - **objects[i].x** will become objects[1].x // this means we got the x coordinate of the second object.
- **Then i is incremented and i =2**
  - The **length** of the array is 2, means the loop terminates.

15. Now let's pass this fetched **x coordinate** inside the **text()** function.

```
text(objects[i].label + " " + percent + "%", objects[i].x
```

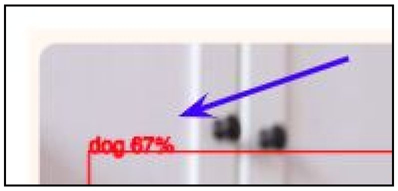
Fetching y coordinates and passing inside text() function

Code for reading confidence from objects array - `objects[i].y`

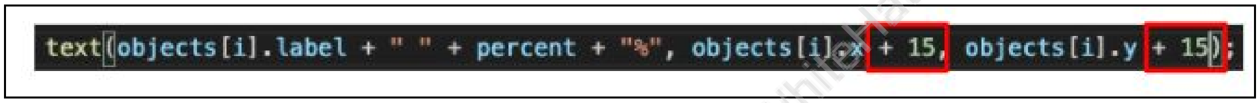
16. Now let's pass this fetched **y coordinate** inside the **text()** function.

```
text(objects[i].label + " " + percent + "%", objects[i].x, objects[i].y);
```

- If we put this code and when we complete the code the rectangle and the label will come on the canvas like this:

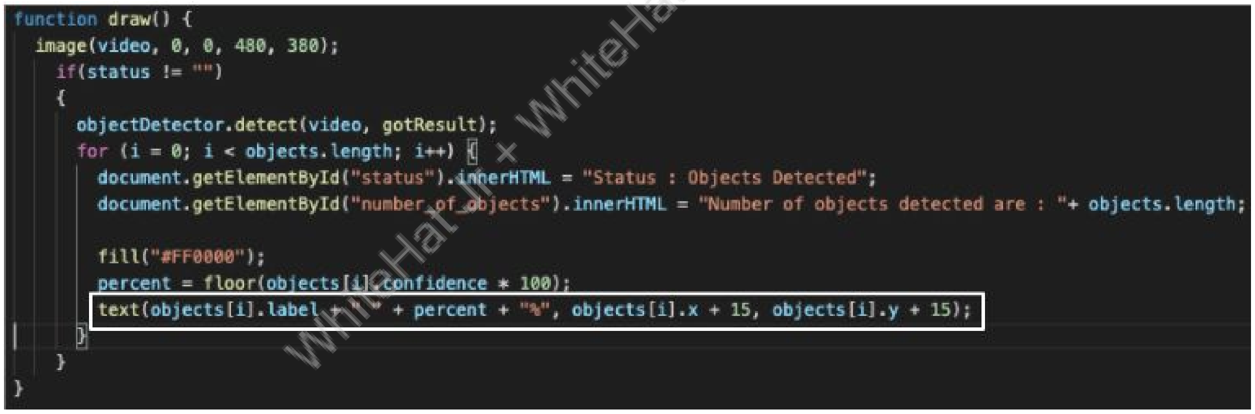


- This will happen because we will be placing **text()** and **rect()** using the same x and y coordinates.
- The solution is we increase x and y coordinates by 15 and then pass it inside the **text()** function. This way we are changing the position of the text.



```
text(objects[i].label + " " + percent + "%", objects[i].x + 15, objects[i].y + 15);
```

Final code for adding label:

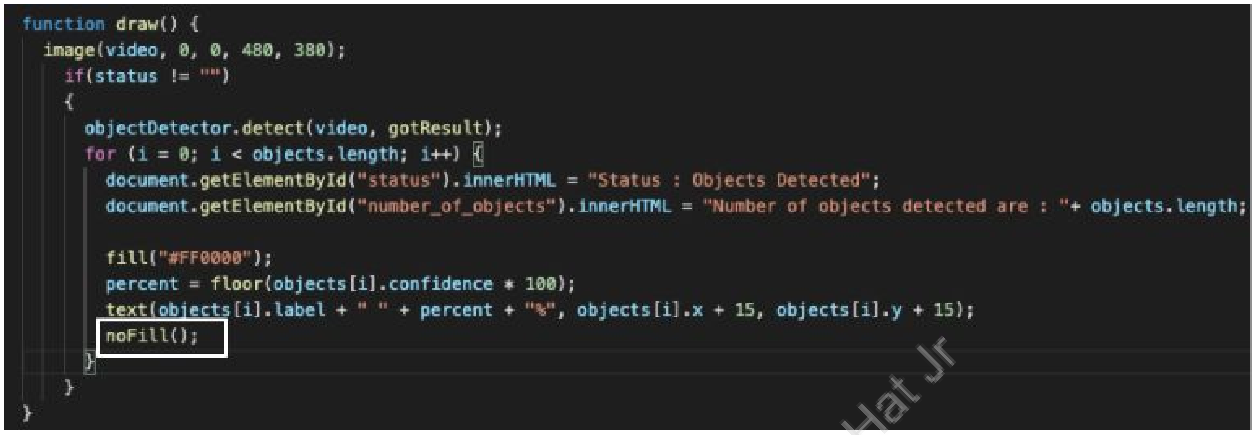


```
function draw() {
  image(video, 0, 0, 480, 380);
    if(status != "")
    {
      objectDetector.detect(video, gotResult);
      for (i = 0; i < objects.length; i++) {
        document.getElementById("status").innerHTML = "Status : Objects Detected";
        document.getElementById("number_of_objects").innerHTML = "Number of objects detected are : "+ objects.length;

        fill("#FF0000");
        percent = floor(objects[i].confidence * 100);
        text(objects[i].label + " " + percent + "%", objects[i].x + 15, objects[i].y + 15);
      }
    }
}
```

17. If we don't unset the color set by **fill()** function then get a weird all rectangles drawn with the color. p5.js **noFill()** function is used to unset the color set by **fill()** function.

Code:

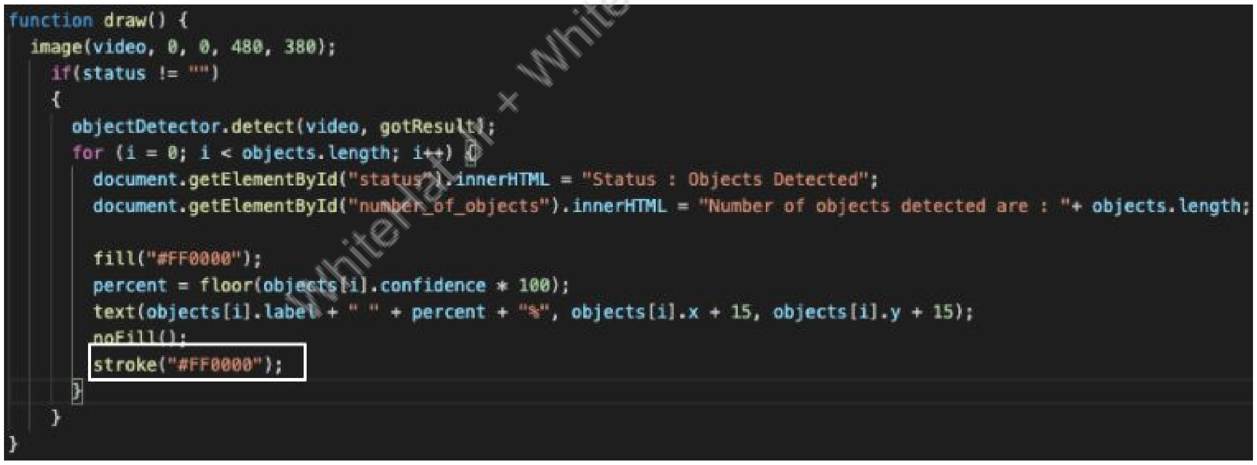


```
function draw() {
  image(video, 0, 0, 480, 380);
    if(status != "")
    {
      objectDetector.detect(video, gotResult);
      for (i = 0; i < objects.length; i++) {
        document.getElementById("status").innerHTML = "Status : Objects Detected";
        document.getElementById("number_of_objects").innerHTML = "Number of objects detected are : "+ objects.length;

        fill("#FF0000");
        percent = floor(objects[i].confidence * 100);
        text(objects[i].label + " " + percent + "%", objects[i].x + 15, objects[i].y + 15);
        noFill();
      }
    }
}
```

18. Now we have to set a border color for the rectangle. **stroke()** function is used for setting the border-color. Inside **stroke()** we can pass HEX code of the color, RGB, or RGBA. So pass HEX code for the red color inside the **stroke()** function.

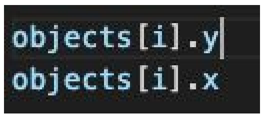
Code:



```
function draw() {
  image(video, 0, 0, 480, 380);
    if(status != "")
    {
      objectDetector.detect(video, gotResult);
      for (i = 0; i < objects.length; i++) {
        document.getElementById("status").innerHTML = "Status : Objects Detected";
        document.getElementById("number_of_objects").innerHTML = "Number of objects detected are : "+ objects.length;

        fill("#FF0000");
        percent = floor(objects[i].confidence * 100);
        text(objects[i].label + " " + percent + "%", objects[i].x + 15, objects[i].y + 15);
        noFill();
        stroke("#FF0000");
      }
    }
}
```

19. Now we can draw a rectangle, for which use a p5.js function **rect()**. Syntax for **rect()** - **rect(x coordinate, y coordinate, width, height).**
20. We already know the code to fetch x and y coordinates from the **objects** array in the for loop.



```
objects[i].y
objects[i].x
```

- So we can pass this fetched x and y coordinate code inside **rect()** function:

```
rect(objects[i].x, objects[i].y,
```

Fetching width and passing inside rect() function

Code to fetch width from objects array `objects[i].width` because for eg - Let's consider there are 2 arrays inside the **objects** array. Means the **length** of this array is 2.

- **When the loop starts i = 0**
    - **objects[i].width** will become objects[0].width // this means we got the width of the first object.
- **Then i is incremented and i = 1**
    - **objects[i].width** will become objects[1].width // this means we got the width of the second object.
- **Then i is incremented and i =2**
    - The **length** of the array is 2, means the loop terminates.

21. Now let's pass this fetched **width** inside the **rect()** function.

```
rect(objects[i].x, objects[i].y, objects[i].width,
```

Fetching height and passing inside rect() function

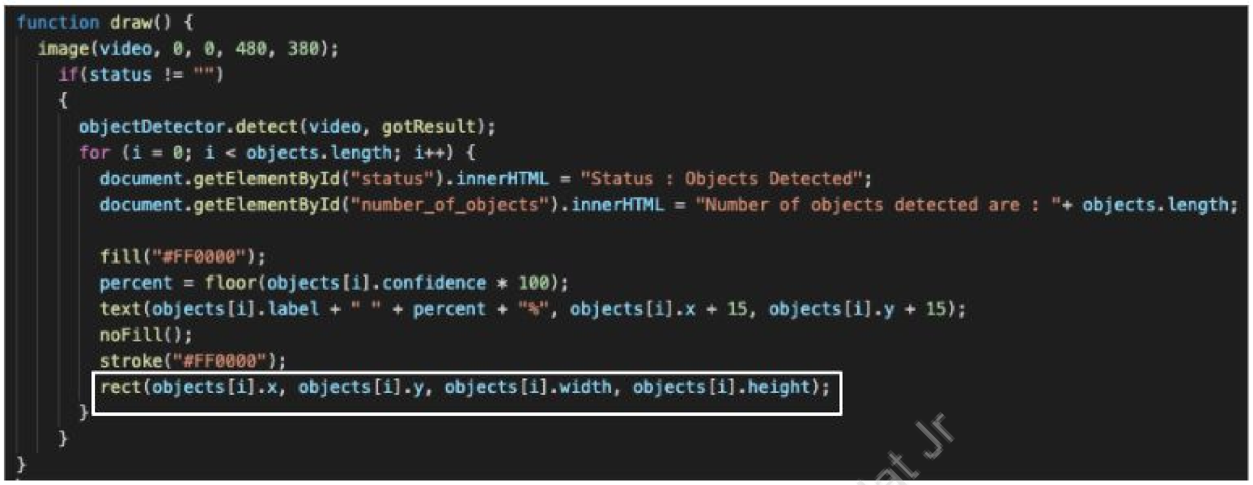
Code to fetch width from objects array `objects[i].height` because for eg - Let's consider there are 2 arrays inside the **objects** array. Means the **length** of this array is 2.

- **When the loop starts i = 0**
    - **objects[i].height** will become objects[0].height // this means we got the height of the first object.
- **Then i is incremented and i = 1**
    - **objects[i].height** will become objects[1].height // this means we got the height of the second object.
- **Then i is incremented and i =2**
    - The **length** of the array is 2, means the loop terminates.

22. Now let's pass this fetched **height** inside the **rect()** function.

```
rect(objects[i].x, objects[i].y, objects[i].width, objects[i].height);
```

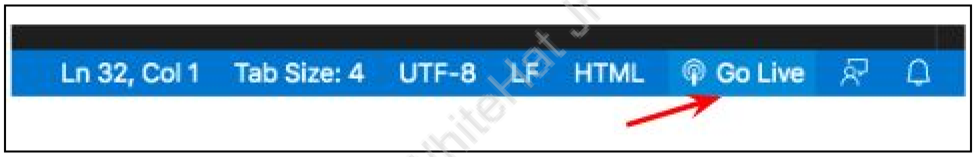
Code:



```
function draw() {
  image(video, 0, 0, 480, 380);
    if(status != "")
    {
      objectDetector.detect(video, gotResult);
      for (i = 0; i < objects.length; i++) {
        document.getElementById("status").innerHTML = "Status : Objects Detected";
        document.getElementById("number_of_objects").innerHTML = "Number of objects detected are : "+ objects.length;

        fill("#FF0000");
        percent = floor(objects[i].confidence * 100);
        text(objects[i].label + " " + percent + "%", objects[i].x + 15, objects[i].y + 15);
        noFill();
        stroke("#FF0000");
        rect(objects[i].x, objects[i].y, objects[i].width, objects[i].height);
      }
    }
}
```

NOTE -
Whenever you are running the code, MAKE SURE TO TEST BY CLICKING ON GO LIVE BUTTON OF VISUAL STUDIO. THIS WILL RESULT IN RUNNING THE FILE ON THE LIVE SERVER



OF VISUAL STUDIO.

Because we are using a video file, and p5.js just doesn't allow us to run any video file from a local system, it needs to be run on a server.

What's NEXT?

We will build an AI Game.