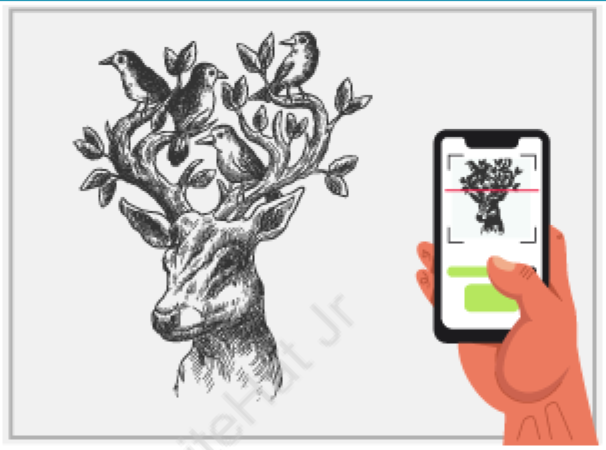
ADV-C129

SKETCH IDENTIFICATION



What was our GOAL for this MODULE?

The goal was to venture into the sea of pretrained neural networks and study them for their possible applications of interest. Learning the neural network methods builds confidence to sift through complex code and apply only that which is needed to solve a problem. Handling complexity is essential in AI.

What did we ACHIEVE in the class TODAY?

- We completed the sketch identification webapp by adding javascript code for getting the sketch from canvas and comparing it with the doodlenet model and getting the result of it.

Which CONCEPTS/ CODING did we cover today?

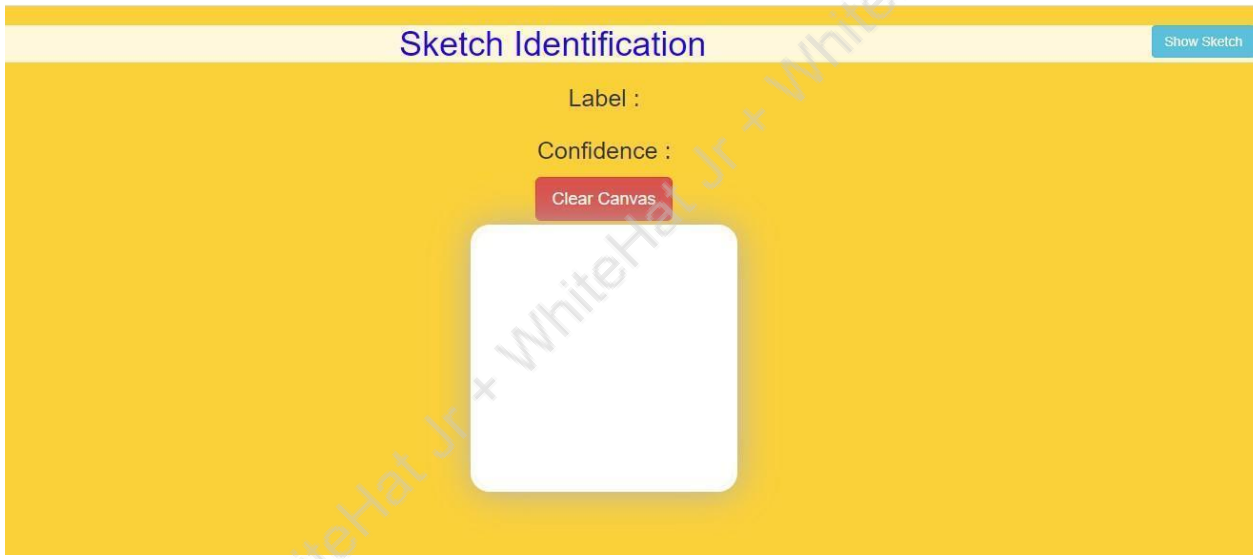
- Added js code for the draw() function to draw the sketches on the canvas.
- Added the preload function to load the model.
- Added the classify() function to classify the sketch.
- Added the gotResult() function to display the sketch name and confidence.

How did we DO the activities?

Code & Explanation

We added the below code in the main.js file.

In the previous classes C127-C128, we added the HTML and CSS code to our application and the output we achieved so far looked like this:

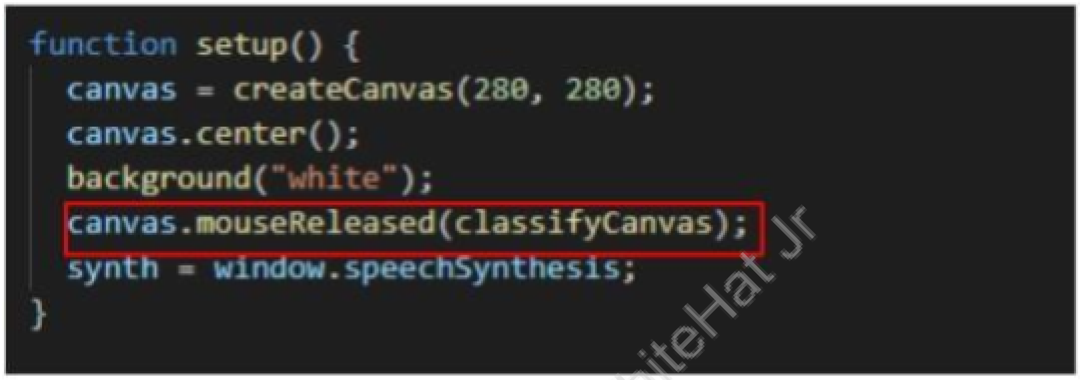


We added the below listed functionalities to our application:

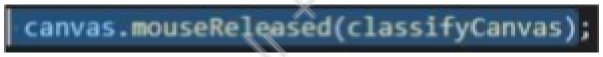
- ➔ Added code to the **setup()** function.
- ➔ Added code to the **preload()** function
- ➔ Added code to the **draw()** function
- ➔ Added code to the **classifycanvas()** function
- ➔ Added code to the **gotResult()** function
- ➔ Added code for the **Text-to-speech** functionality

1. Added code to the setup() function: We added some code to the setup() function which we have created in the previous class.

- So, we added the below code: are calling the function **classifyCanvas()** which

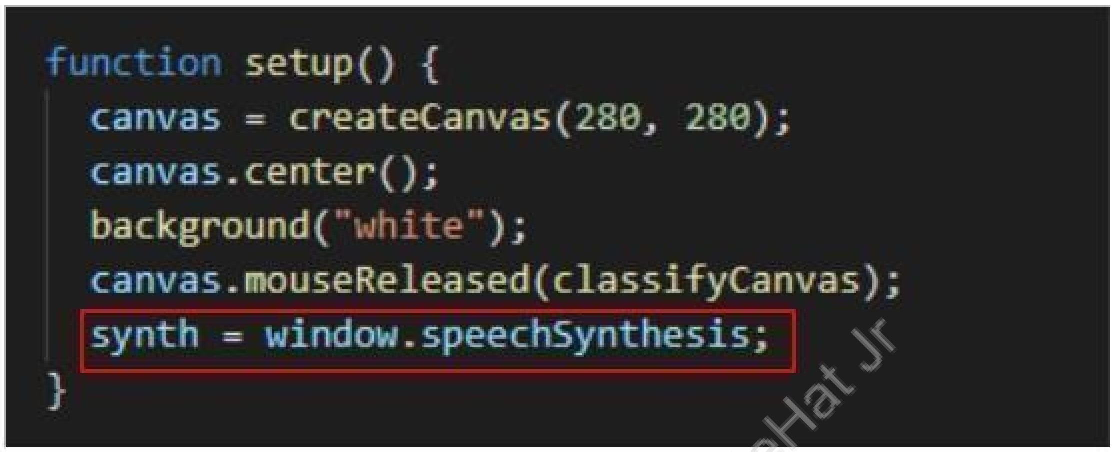


```
function setup() {
  canvas = createCanvas(280, 280);
  canvas.center();
  background("white");
  canvas.mouseReleased(classifyCanvas);
  synth = window.speechSynthesis;
}
```

By using this code  `canvas.mouseReleased(classifyCanvas);`, basically, we we defined later in the class. So, this function gets triggered when the user

clicks on the canvas and releases on the mouse click.

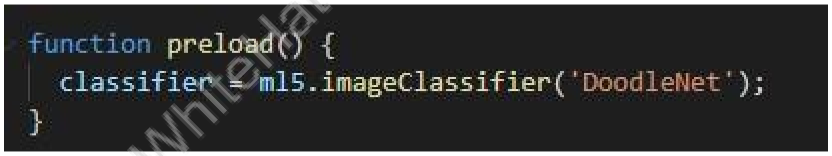
- Then, we added the below code:



```
function setup() {
  canvas = createCanvas(280, 280);
  canvas.center();
  background("white");
  canvas.mouseReleased(classifyCanvas);
  synth = window.speechSynthesis;
}
```

We added the code `synth = window.speechSynthesis;`. By doing this, we are initializing the available devices or the synthesis on the system. We saw in the demo that after the sketch was identified, the system was speaking out the sketch name, hence we needed the text-to-speech functionality. That's why we defined the **speechSynthesis** API and stored it in the **synth** variable.

2. **Created the preload() function:** Next, after this, we created a function named preload().

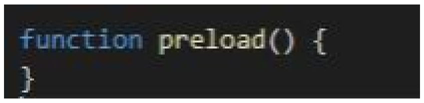


```
function preload() {
  classifier = ml5.imageClassifier('DoodleNet');
}
```

We needed to load the doodlenet model, and called the **ml5.imageClassifier()** function to check whether the model was loaded.

Explaining the above line of code in detail:

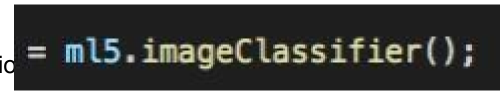
➔ First we created a function named preload():



```
function preload() {
}
```

```
classifier =
```

```
= ml5.
```



```
= ml5.imageClassifier();
```

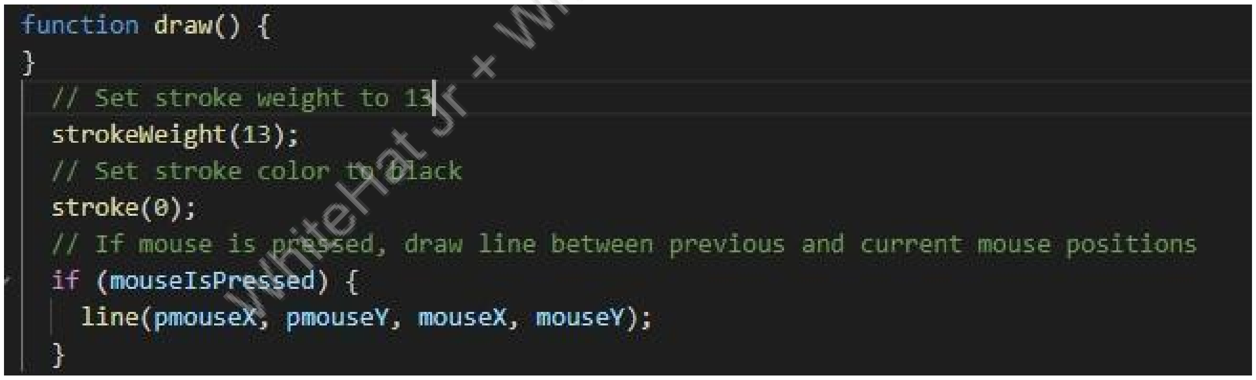
- ➔ Then, inside this function, we defined a variable ➔ Then, we wrote the library

  name which is **ml5**.js:

- ➔ Then, wrote imageClassifier, like this:
  - **imageClassifier** is a predefined function of ml5.js that is used to trigger the ml5.js image classification function.
- ➔ Then, we passed one parameter which is the name of the model we used, which was the 'DoodleNet'.

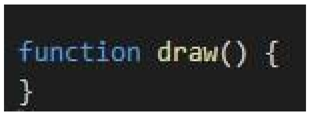
We can put this **ml5.imageClassifier()** function inside the setup() function, as we had done in previous classes. We added it in the **preload()** function, just to show that the model initialization code can be added in the preload() function as well.

3. **Created a draw() function:** Then, to be able to draw a sketch on the canvas, we needed to set up the functionality of what will be the stroke thickness, stroke color etc.
   So, we created a draw() function like this:



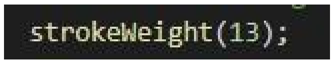
```
function draw() {
}
  // Set stroke weight to 13
  strokeWeight(13);
  // Set stroke color to black
  stroke(0);
  // If mouse is pressed, draw line between previous and current mouse positions
  if (mouseIsPressed) {
    line(pmouseX, pmouseY, mouseX, mouseY);
  }
```

- ➔ So, first, we created a draw() function:



```
function draw() {
}
```

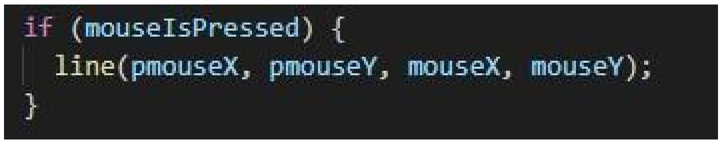
- ➔ And inside this function, we set the weight of the stroke to draw any image:



```
strokeWeight(13);
```

➔ Then, we set the default color of the stroke as black. So whatever sketch we draw, was in black color: `stroke(0);`.

➔ Then, to draw any sketch, we added a condition that, if we press the mouse, then the line should be drawn from the previous mouse position to the current mouse position, So, we added this code:

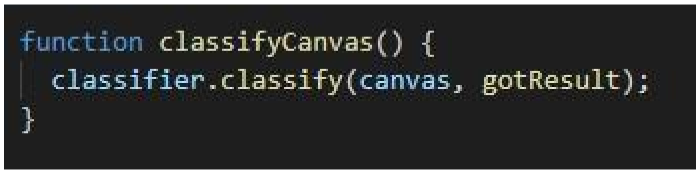


```
if (mouseIsPressed) {
  line(pmouseX, pmouseY, mouseX, mouseY);
}
```

- So, here, inside the if statement, we wrote **mouseIsPressed** which is a boolean system variable, which returns *true* if the mouse was pressed, and it returns *false* if the mouse is not pressed.
- Then, we used the **line()** function, which is used to draw lines onto the canvas. It takes 4 parameters. Which are -
    - **pmouseX** - This is the p5.js variable which always contains the previous horizontal mouse X coordinate on the canvas.
    - **pmouseY** - This is the p5.js variable which always contains the previous vertical mouse Y coordinate on the canvas.
    - **mouseX** - This is the p5.js variable that always contains the current horizontal mouse X coordinate on the canvas.
    - **mouseY** - This is the p5.js variable which always contains the current vertical mouse Y coordinate on the canvas. So, using these 4 values, we draw on canvas.

4. **Defined the classifyCanvas() function:** Then, we defined the classifyCanvas function. This is the same function which we had called in point 1 inside the setup() function on mouse release.

The functionality of this function was to compare sketches with the model, and get the results from the model.

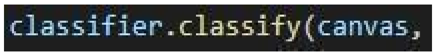


```
function classifyCanvas() {
  classifier.classify(canvas, gotResult);
}
```

`classifier.`riable that holds the model - doodlenet model.

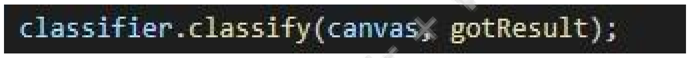
- `classify` is a predefined function of ml5.js which is used to compare images with the model, and get the results. In `classify(` function, we need to pass two things:

1) input - That is **canvas** as we got the input from the canvas
.
2) gotResult- This function gets called when the prediction is made, and inside which we were able to access the result of the prediction. This will hold the result of the comparison. We will define this function in the next step.



```
classifier.classify(canvas,
```

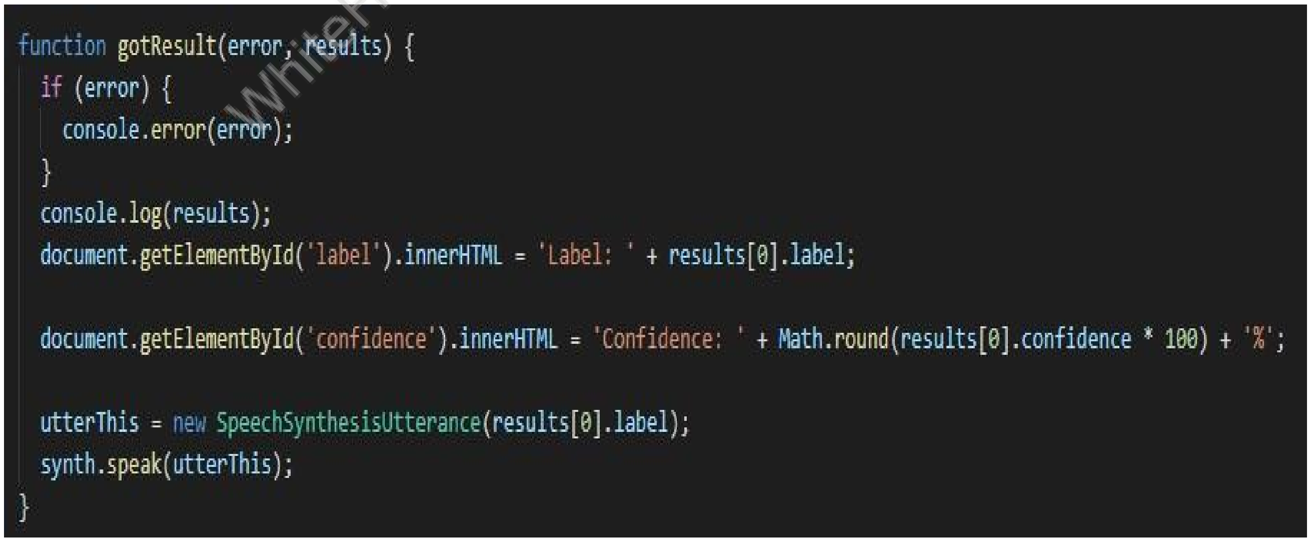
- The **canvas** variable contains the canvas.
- Second a function, which holds the result of the comparison.



```
classifier.classify(canvas, gotResult);
```

- **gotResult** function holds the result of the comparison. We defined this function in the next step.

5. Code for the gotResult() function: We Wrote code for the **gotResult()** function.



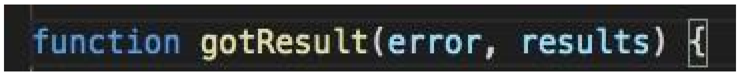
```
function gotResult(error, results) {
  if (error) {
    console.error(error);
  }
  console.log(results);
  document.getElementById('label').innerHTML = 'Label: ' + results[0].label;

  document.getElementById('confidence').innerHTML = 'Confidence: ' + Math.round(results[0].confidence * 100) + '%';

  utterThis = new SpeechSynthesisUtterance(results[0].label);
  synth.speak(utterThis);
}
```

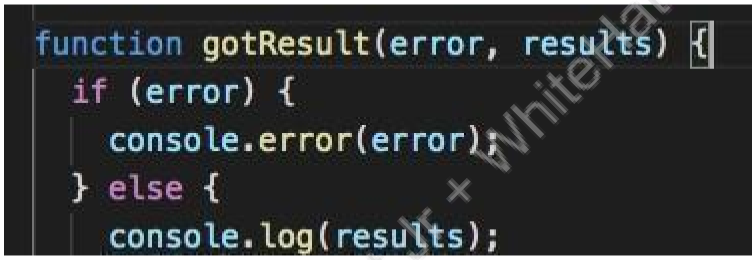
This **gotResult** function holds the result of the comparison and has two parameters inside it - the first one is **error** and the second one is **results**.

➔ So when we defined the **gotResult** function, we needed to pass **error** and **results** inside the function.



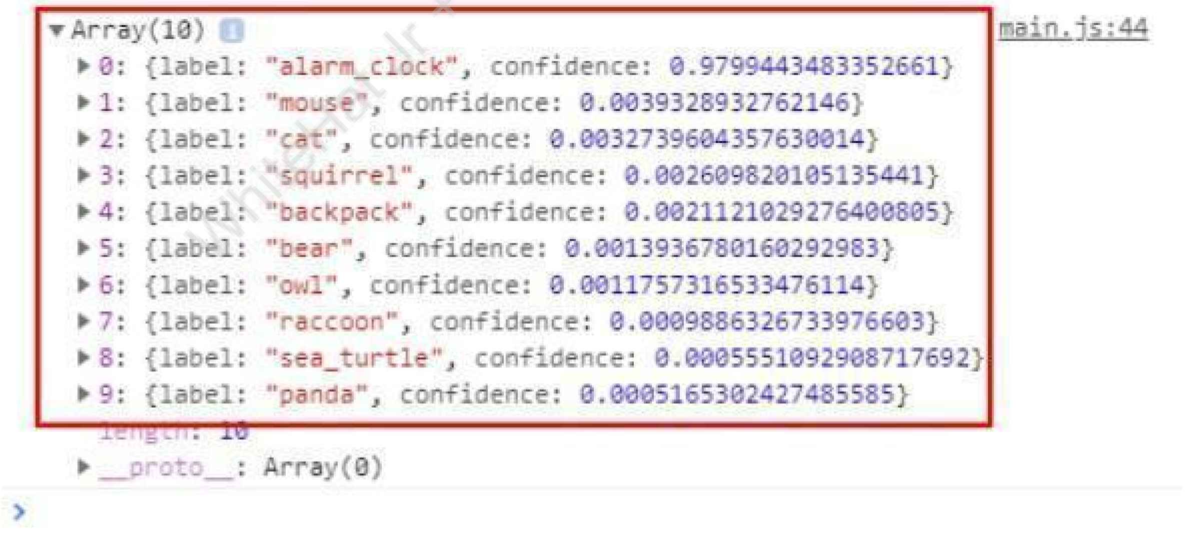
```
function gotResult(error, results) {
```

➔ Then, we checked if there was an error. If yes, then we needed to console the error, else we had to console the results, and see what the results are.



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

- Output on console screen:



```
▼Array(10)                                                          main.js:44
  ▶0: {label: "alarm_clock", confidence: 0.9799443483352661}
  ▶1: {label: "mouse", confidence: 0.003932893276214б}
  ▶2: {label: "cat", confidence: 0.0032739604357630014}
  ▶3: {label: "squirrel", confidence: 0.002609820105135441}
  ▶4: {label: "backpack", confidence: 0.0021121029276400805}
  ▶5: {label: "bear", confidence: 0.0013936780160292983}
  ▶6: {label: "owl", confidence: 0.0011757316533476114}
  ▶7: {label: "raccoon", confidence: 0.0009886326733976603}
  ▶8: {label: "sea_turtle", confidence: 0.0005551092908717692}
  ▶9: {label: "panda", confidence: 0.0005165302427485585}
   length: 10
  ▶__proto__: Array(0)
>
```

As we know that the first result is the most accurate one, we fetched the first result label and its confidence and printed it on the HTML page.

To get the label from the first result -

- The array which we expanded in the console screen is result, which means that we wrote **results** first.
- Then, we wanted the first label, and it is inside the 0th index -
  `▸0: {label: "alarm_clock", confidence: 0.9799443483352661}` . It means that we wanted the 0th index which is inside the **results.** So we wrote **results[0]**.
- Now, we wanted the **label** which is inside the 0th index. So we wrote **results[0].label.**

We knew how to get the value of the first label which is inside the result. So, we updated the HTML element which we defined to hold the object name in **index.html** with **results[0].label**.

- Code:

```
document.getElementById('label').innerHTML = 'Label: ' + results[0].label;
```

To get the confidence from the first result -

- The array which we expanded in the console screen is results, which means that first we wrote **results**.

- We wanted the first confidence, and it is inside at 0th index -
  `▸0: {label: "alarm_clock", confidence: 0.9799443483352661}` . It means that we want the 0th index which is inside the **results.** So, we wrote **results[0].**
- We wanted the **confidence** which is inside the 0th index. So, we wrote **results[0].confidence.**

We knew how to get the value of the first **confidence** which is inside the result. So we updated the HTML element which we defined to hold the accuracy of the decision in **index.html** with **result[0].confidence.**

```
document.getElementById('confidence').innerHTML = 'Confidence: '
```

Now, the value of the confidence is given as a float value which is `confidence: 0.9257816076278687}` , and we needed to show confidence in percentage. so, we had to convert the float to percentage. So, we will write this code:

```
document.getElementById('confidence').innerHTML = 'Confidence: ' + Math.round(results[0].confidence * 100) + '%';
```

- So here, we have used the **math.round()** function. Basically, it is used to round off the value to the nearest integer. So, we are rounding off the confidence value to the nearest integer `Math.round(results[0].confidence` and then converting it to percentage by multiplying it to 100. Like this:

```
Math.round(results[0].confidence * 100)
```

- And then we concatenate the '%' sign to the value.

```
Math.round(results[0].confidence * 100) + '%'
```

For example, if we consider a float value as 0.82345, and if we convert it to percentage, we will multiply it by 100.
0.82345 * 100 = 82.345
Now, to round off this value, this means to convert the float onto percentage - we will pass this value in Math.round() function like this:

Math.round(**82.345**) + '%' = **82%**

Then, as we wanted our application to speak what the name of the object being sketched, for this, we wrote the code as:

```
utterThis = new SpeechSynthesisUtterance(results[0].label);
```

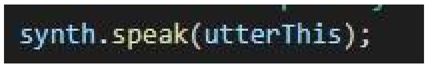
So here,

- **utterThis** - is a variable in which we stored the converted text to speech.
- **SpeechSynthesisUtterance** - is the function of an API that converts text to speech.
- We are using a **new** keyword because, for every next text, we wanted to convert that text to speech.
- **results[0].label**- contains the text that is the name of the object drawn.

We created a variable `utterThis`, and by using the **SpeechSynthesisUtterance()**, we passed **results[0].label**, which holds the name of the object being sketched on the canvas.

Then, we converted text to speech and stored it inside a variable. So, we passed this variable to the speak() function of the API.

Hence, we passed the utter variable, which holds the accurate name of the object being drawn to the **synth.speak()** function like this so that it gives the output in speech form:



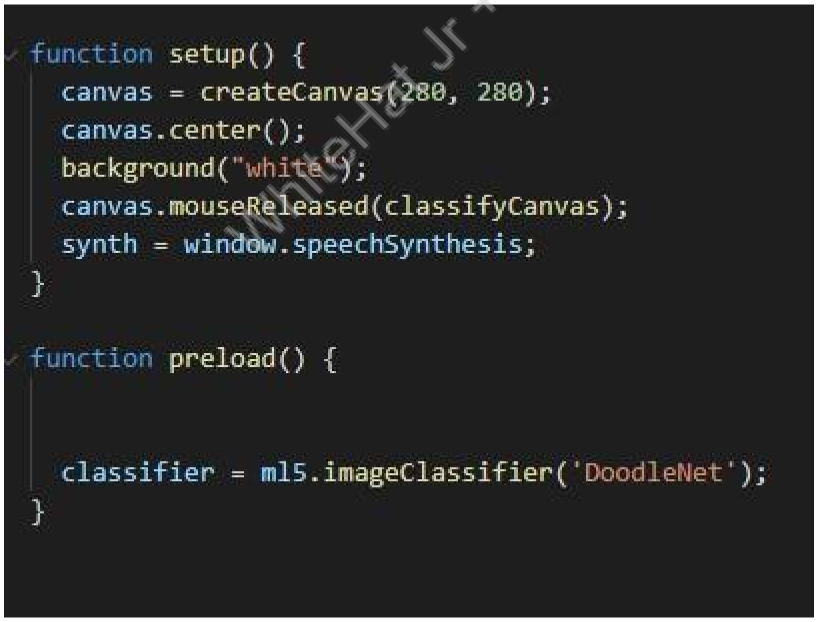
```
synth.speak(utterThis);
```

So here,

- **synth** - In this, we stored the API in point 1.
- speak() - speak() function is a predefined function of the API.
- **utterThis** - holds the converted value of text to speech that we want the system to speak.

The functionality of this speak() function is to trigger the system to speak whatever is passed inside this speak function.

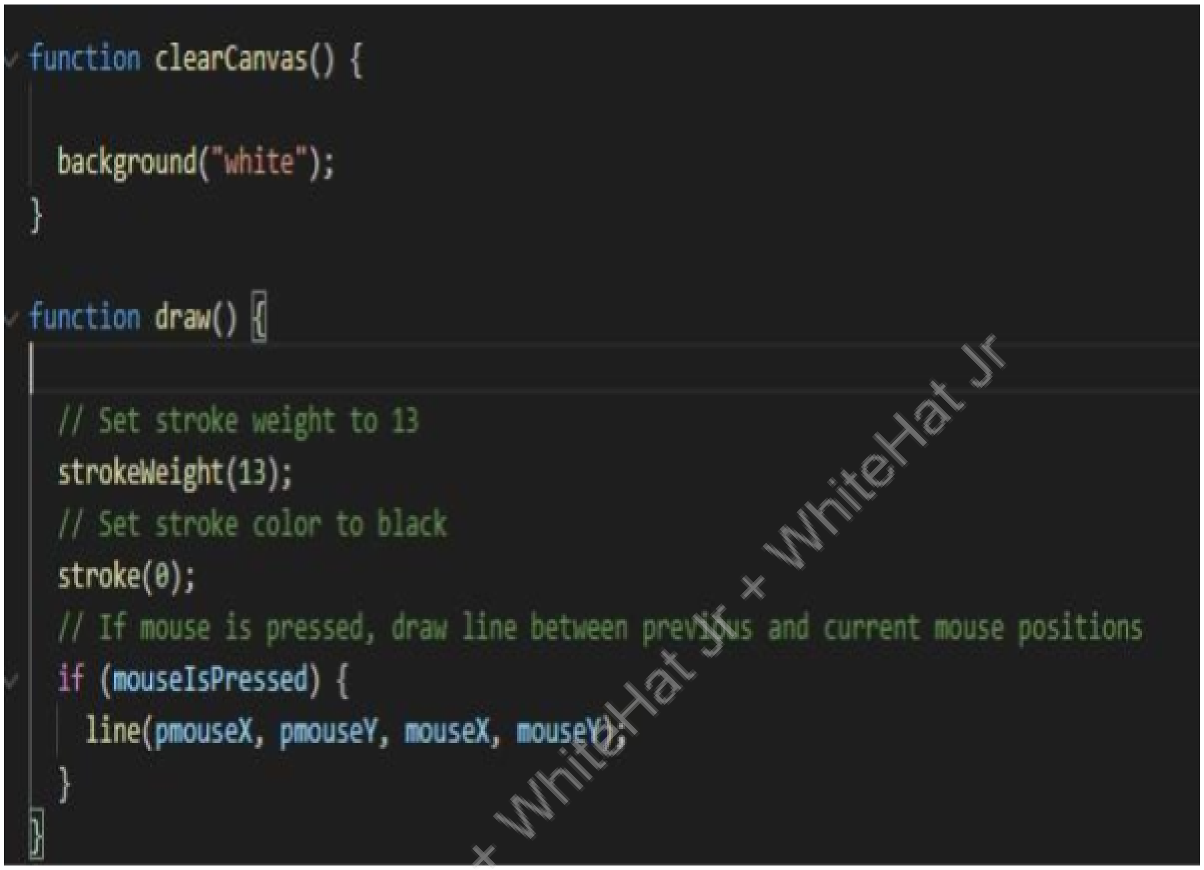
Full code:



```
function setup() {
  canvas = createCanvas(280, 280);
  canvas.center();
  background("white");
  canvas.mouseReleased(classifyCanvas);
  synth = window.speechSynthesis;
}

function preload() {


  classifier = ml5.imageClassifier('DoodleNet');
}
```

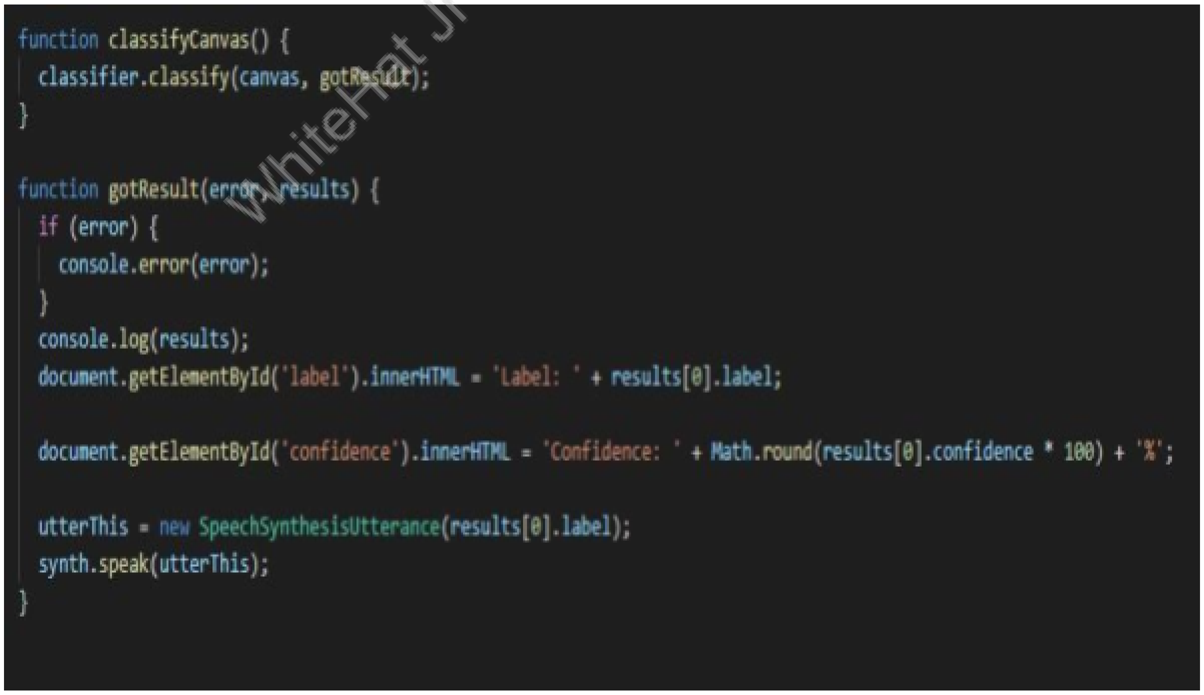


```
function clearCanvas() {

  background("white");
}

function draw() {

  // Set stroke weight to 13
  strokeWeight(13);
  // Set stroke color to black
  stroke(0);
  // If mouse is pressed, draw line between previous and current mouse positions
  if (mouseIsPressed) {
    line(pmouseX, pmouseY, mouseX, mouseY);
  }
}
```



```
function classifyCanvas() {
  classifier.classify(canvas, gotResult);
}

function gotResult(error, results) {
  if (error) {
    console.error(error);
  }
  console.log(results);
  document.getElementById('label').innerHTML = 'Label: ' + results[0].label;

  document.getElementById('confidence').innerHTML = 'Confidence: ' + Math.round(results[0].confidence * 100) + '%';

  utterThis = new SpeechSynthesisUtterance(results[0].label);
  synth.speak(utterThis);
}
```

What’s NEXT?

In the next class, we will be creating the HTML structure for Real Time Image Identification web app which will be using Mobilenet Model.