

## NATURAL LANGUAGE PROCESSING



### What is our GOAL for this MODULE?

We started building our voice controlled selfie web app. We learned to use the speech to text API in our JS code as per our need.

### What did we ACHIEVE in the class TODAY?

- Converted speech to text.
- Learned about the **new** keyword.

### Which CONCEPTS/ CODING did we cover today?

- Completed the HTML code for the **voice\_selfie\_app.html** file.
- Wrote the JS code for converting speech to text.

### How did we DO the activities?

The **Voice\_selfie\_app** folder has:

- **voice\_selfie\_app.html** file - it has some prewritten HTML code.
- **style.css** - it has all the CSS properties, no need to touch this file.
- **main.js** file - it is a blank JS file.

voice\_selfie\_app.html code:

```

<html>
  <head>
    <title>My Selfie App</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>

    <link href="style.css" rel="stylesheet">
  </head>

  <body>
    <div class="container">
      <center>
        <h1>Voice Selfie App
        </h1>
        <hr>
        <h4 style="color:red;">Please press allow as soon as popup comes. After you give a voice command</h4>

        <div class="form-group">
          <h3>Say "take my selfie"</h3>
          <label>Your voice output : </label>

          <p>Press the Start button</p>

          <label>Your webcam : </label>

          <br>

          <label>Your selfie Will Display Here: </label>

        </div>
      </center>
    </div>

    <script src="main.js"></script>
  </body>
</html>

```

Bootstrap link

Our stylesheet link

Our JS link

This HTML file has:

- **Bootstrap** links : To make the app compatible with phone screens as we all take selfies from our phones. So we want our app to work on phone screens as well.
- our **style.css** file link
- our **main.js** file link
- and some html codes which you already know

Now let's add some important HTML elements inside this file:

1. Add an **anchor** tag which will be used to download the selfie:

```
<body>
  <div class="container">
    <center>
      <a href="" id="link" download="myselfie.png"></a>
```

- we are keeping **href=""** empty because we will update this with the selfie link from javascript in the upcoming classes.
- **id="link"** - this id will be used for identifying this anchor tag using JS while updating its href value.
- **download="myselfie.png"** - the selfie will be saved with the name as **myselfie.png**, if you want the selfie to be downloaded with any other name, assign that name to download.
- And we are not writing any text for this anchor tag here:

```
<body>
  <div class="container">
    <center>
      <a href="" id="link" download="myselfie.png"></a>
```

- Because we don't want it to be displayed on the page. Use this anchor tag from JS to download the selfie.

2. Replace the image in the image tag with your image.

```
<h1>Voice Selfie App

</h1>
```

- Now add a textarea, it will be used to hold the text that will be generated after it is converted from our audio.

```
<div class="form-group">
<h3>Say "take my selfie"</h3>
<label>Your voice output : </label>
<textarea class="form-control" id="textbox" readonly="true"></textarea>
```

- **<textarea> tag** - defines a multi-line text input control.
- **class="form-control"** - its a bootstrap class it adds the padding and margin to the textarea.
- **id="textbox"** - This id will be used in JS code for updating the value of this textarea.
- **readonly="true"** - If we set this to true, it means it won't allow the user to type anything inside this textarea. We are setting it true because we don't want anyone to edit the values which are generated inside this textarea.
- **Output:**



- Now add a start button:

```
<button class="btn btn-primary" onclick="start()">Start</button>
<p>Press the Start button</p>
```

- **button tag** - we already know about it.
- **class="btn btn-primary"** - these are bootstrap classes:
  - **btn** - This will remove the default style of the button.
  - **btn-primary** - This will add dark blue background color and dark blue

border color to the button.

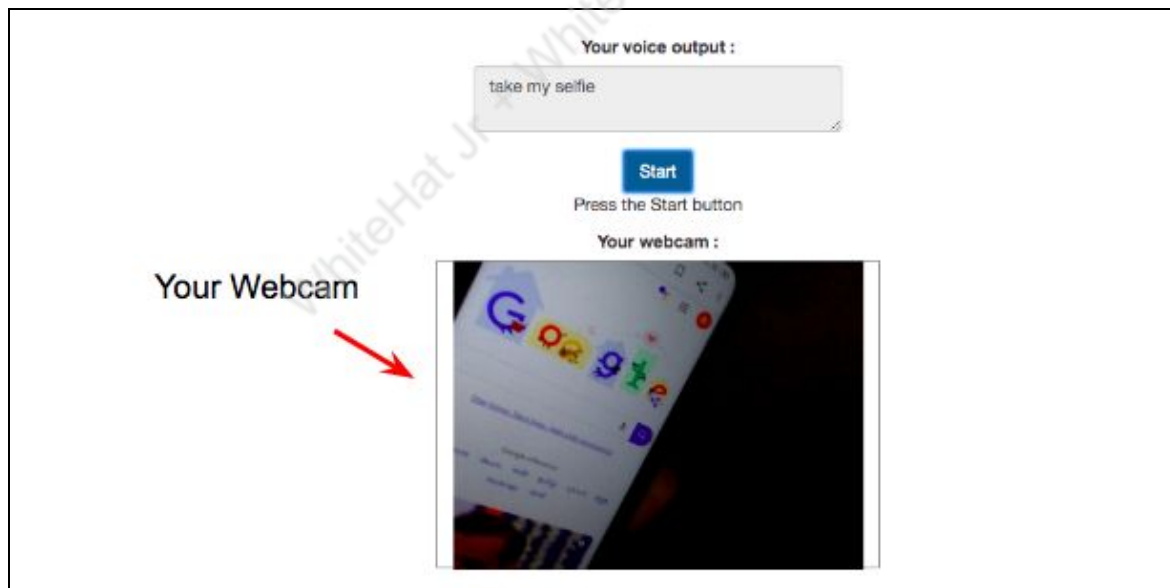
- **onclick="start()"** - when the button is clicked it will call this **start()** function. The functionality of the start function is, it will start recording our voice and convert it to text.
- **Output:**



5. Now add a div that will be used to reflect the webcam output:

```
<label>Your webcam : </label>  
<div id="camera"></div>
```

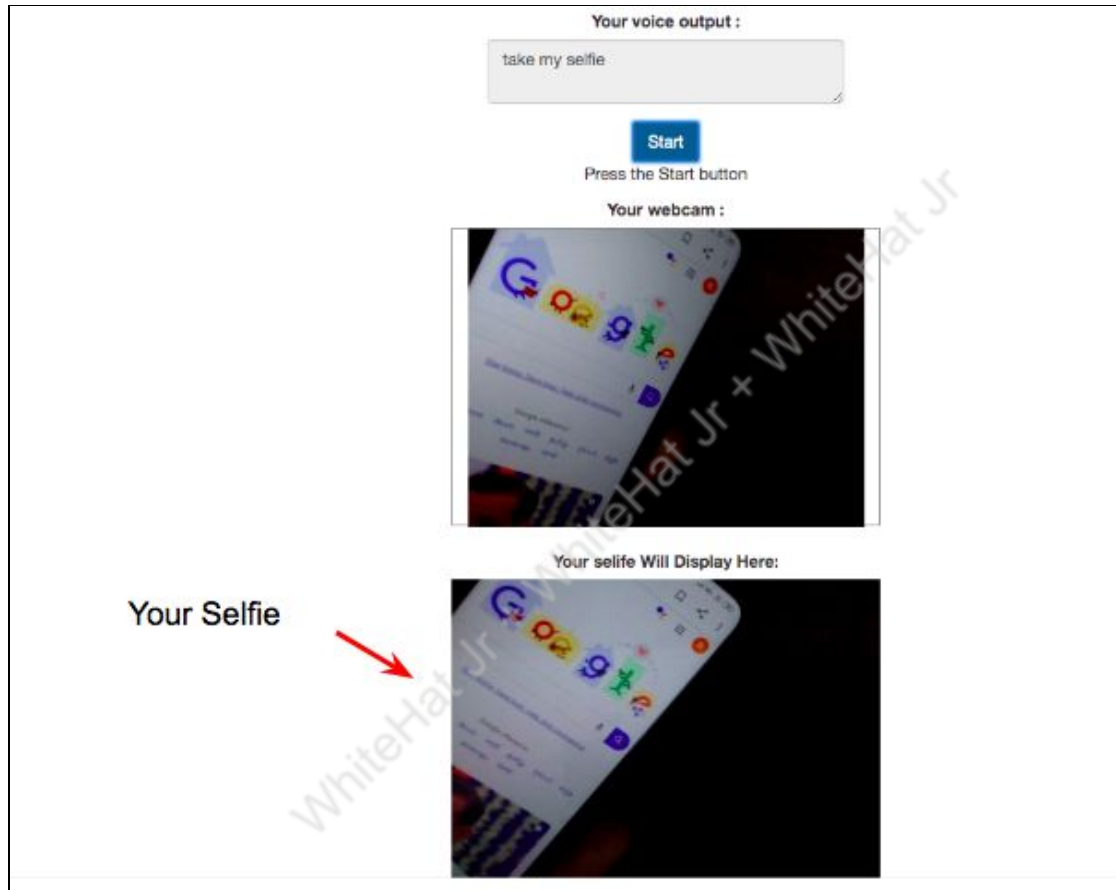
- The id given to this div will be used for identifying this div, while updating this div with the webcam output.
- **Output:**



6. Now add a div that will be used to hold the selfie taken by the webcam:

```
<label>Your selfie Will Display Here: </label>  
<div id="result" ></div>
```

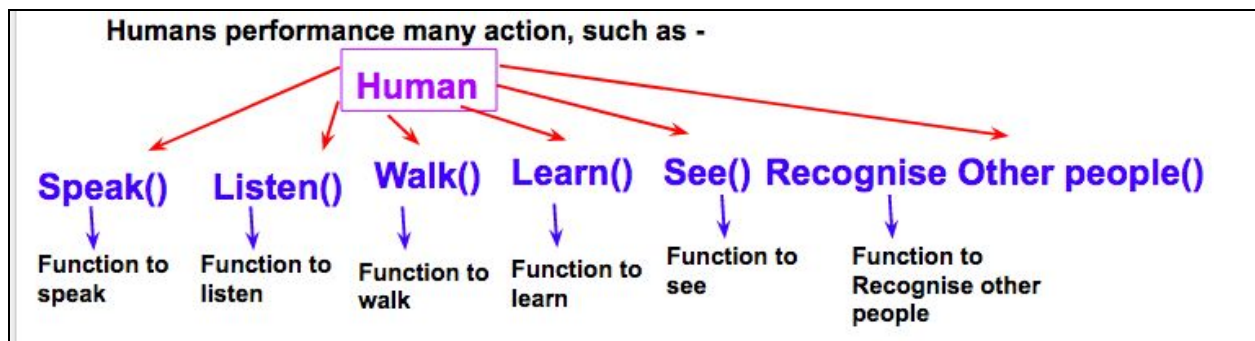
- The id given to this div will be used for identifying this div, while updating this div with the selfie taken by the webcam.
- Output:



Now write the JS code in the **main.js** file which you have downloaded.

Before explaining the JS code understand the concept of the **new** keyword and why it is used.





So if we want to create a new human with a name john, and ask him to speak.

So from the above image we can understand that **human** has all function to perform all actions. So for understanding let's say this **human** is an API and to create new human everytime we will use this API

Creating Human steps -

1. First we will create a new human with name John.

**John = new human()**

 Name of the human
  **New** keyword is use to create new human
  Calling the **human** API, as per the above explanation

2. And ask the john to speak

**John.speak()**

 Name of the human
  Function to perform action speak

So the new keyword is used for reusing anything again that was created earlier.

JS code for Speech to Text:  
main.js file:

```
var SpeechRecognition = window.webkitSpeechRecognition;

var recognition = new SpeechRecognition();

function start()
{
  document.getElementById("textbox").innerHTML = "";
  recognition.start();
}

recognition.onresult = function run (event) {

  console.log(event);

  var Content = event.results[0][0].transcript;
  console.log(Content);

  document.getElementById("textbox").innerHTML = Content;
}
```

Explaining the above code in pieces:



```
var SpeechRecognition = window.webkitSpeechRecognition;

1. var recognition = new SpeechRecognition();
```

```
window.webkitSpeechRecognition;
```

This is the **Web Speech API** used to recognise what we are speaking and convert it in to text

First we will store this API into a variable, so that we can use this **Web Speech API**, like this -

```
var SpeechRecognition = window.webkitSpeechRecognition;
```

The way we created new human - **John = new human()**

The same way we will create a new **Web Speech API** to use in our webapp, and store it inside a variable

```
var recognition = new SpeechRecognition();
```

Variable, to store new  
**Web Speech API**

Calling the **Web Speech API**, as per the above  
explanation

**New** keyword is use to create **Web Speech API**

2. Now define the start function and write code for it:

```
function start()
{
  document.getElementById("textbox").innerHTML = "";
  recognition.start();
}
```

- Now define the **start()** function:

```
function start()
{
```

- Whenever the start button is pressed we want the textarea to be empty. For that we are updating the textarea with an empty value.

```
document.getElementById("textbox").innerHTML = "";
```

After creating new human with a name john

```
John = new human()
```

We asked john to speak, by calling **speak()** from **human API**

```
John.speak()
```

The same way after creating new **Web Speech API** and storing it inside a variable

```
var recognition = new SpeechRecognition();
```

We will call the **start()** function from **Web Speech API**.

```
recognition.start();
```

**This start function is a predefined function of Web Speech API and it will convert your speech to text.**

3. Explaining the function to display the result on the HTML page. This result is the

conversion of our speech to text(which is done by `recognition.start();` in step 2)

```
recognition.onresult = function(event) {  
  
    console.log(event);  
  
    var Content = event.results[0][0].transcript;  
    console.log(Content);  
  
    document.getElementById("textbox").innerHTML = Content;  
}
```

Explaining the above code:

- We call the **start()** function like this:

```
recognition.start();
```

- The same way call the **onresult** function:

```
recognition.onresult
```

- The **onresult** function holds all the values of speech converted to text.
- So to get these values from **onresult** write a function like this:

```
recognition.onresult = function(event) {
```

- Now console this event and see what it has:

```
recognition.onresult = function(event) {  
  console.log(event);
```

- Output on console screen:

```
main.js:13  
▶ SpeechRecognitionEvent {isTrusted: true, resultIndex: 0, results: Speech  
  RecognitionResultList, interpretation: null, emma: null, ...}
```

- Click on the sign shown in the below image:

```
main.js:13  
▶ SpeechRecognitionEvent {isTrusted: true, resultIndex: 0, results: Speech  
  RecognitionResultList, interpretation: null, emma: null, ...}
```

4. Keep a track on which section is clicked. This is done because we need to know in which section our speech to text output is there. So when we find that section fetch the speech to text output from that section.



```
SpeechRecognitionEvent {isTrusted: true, resultIndex: 0, results: Speech
RecognitionResultList, interpretation: null, emma: null, ...}
  isTrusted: true
  resultIndex: 0
  results: SpeechRecognitionResultList {0: SpeechRecognitionResult, len...
  interpretation: null
  emma: null
  type: "result"
  target: SpeechRecognition {grammars: SpeechGrammarList, lang: "", con...
  currentTarget: SpeechRecognition {grammars: SpeechGrammarList, lang: ...
  eventPhase: 0
  bubbles: false
  cancelable: false
  defaultPrevented: false
  composed: false
  timeStamp: 5615921.1450000005
  srcElement: SpeechRecognition {grammars: SpeechGrammarList, lang: "",...
  returnValue: true
  cancelBubble: false
  path: []
  __proto__: SpeechRecognitionEvent
```

- You will see too much data, but we don't need this data, so click on the sign as per the below image.
5. Keep a track on which section you clicked, in the image below we have clicked on the results, so we can say inside the event we have clicked on the results. In code it will be written as **event.results**.



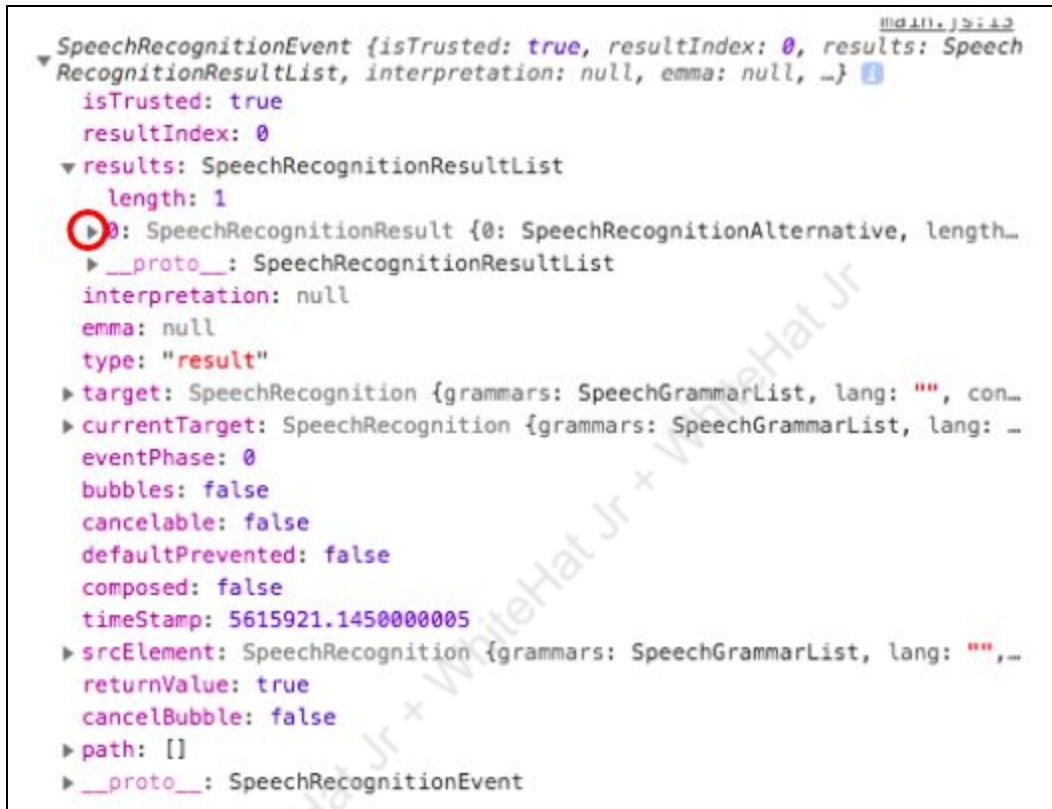
```
SpeechRecognitionEvent {isTrusted: true, resultIndex: 0, results: Speech
RecognitionResultList, interpretation: null, emma: null, ...} ⓘ
  isTrusted: true
  resultIndex: 0
  results: SpeechRecognitionResultList {0: SpeechRecognitionResult, len...
  interpretation: null
  emma: null
  type: "result"
  target: SpeechRecognition {grammars: SpeechGrammarList, lang: "", con...
  currentTarget: SpeechRecognition {grammars: SpeechGrammarList, lang: ...
  eventPhase: 0
  bubbles: false
  cancelable: false
  defaultPrevented: false
  composed: false
  timeStamp: 5615921.1450000005
  srcElement: SpeechRecognition {grammars: SpeechGrammarList, lang: "",...
  returnValue: true
  cancelBubble: false
  path: []
  __proto__: SpeechRecognitionEvent
```

After clicking on results:

```
main.js:13
SpeechRecognitionEvent {isTrusted: true, resultIndex: 0, results: Speech
RecognitionResultList, interpretation: null, emma: null, ...} ⓘ
  isTrusted: true
  resultIndex: 0
  results: SpeechRecognitionResultList
    length: 1
    0: SpeechRecognitionResult {0: SpeechRecognitionAlternative, length...
    __proto__: SpeechRecognitionResultList
  interpretation: null
  emma: null
  type: "result"
  target: SpeechRecognition {grammars: SpeechGrammarList, lang: "", con...
  currentTarget: SpeechRecognition {grammars: SpeechGrammarList, lang: ...
  eventPhase: 0
  bubbles: false
  cancelable: false
  defaultPrevented: false
  composed: false
  timeStamp: 5615921.1450000005
  srcElement: SpeechRecognition {grammars: SpeechGrammarList, lang: "",...
  returnValue: true
  cancelBubble: false
  path: []
  __proto__: SpeechRecognitionEvent
```

- You will see too much data, but we don't need this data, so click on the sign as per the below image.

6. As we are keeping a track on which section we have clicked till now, we have clicked on the results which are inside the event. In code it will be written as **event.results**. In the image below we have clicked on the 0 which is inside the results, so we can say inside the event we have clicked on the results, inside the results we have clicked 0. In code it will be written as **event.results[0]**.



```
SpeechRecognitionEvent {isTrusted: true, resultIndex: 0, results: Speech
RecognitionResultList, interpretation: null, emma: null, ...}
  isTrusted: true
  resultIndex: 0
  results: SpeechRecognitionResultList
    length: 1
    0: SpeechRecognitionResult {0: SpeechRecognitionAlternative, length...
      __proto__: SpeechRecognitionResultList
      interpretation: null
      emma: null
      type: "result"
      target: SpeechRecognition {grammars: SpeechGrammarList, lang: "", con...
      currentTarget: SpeechRecognition {grammars: SpeechGrammarList, lang: ...
      eventPhase: 0
      bubbles: false
      cancelable: false
      defaultPrevented: false
      composed: false
      timeStamp: 5615921.1450000005
      srcElement: SpeechRecognition {grammars: SpeechGrammarList, lang: "",...
      returnValue: true
      cancelBubble: false
      path: []
      __proto__: SpeechRecognitionEvent
```



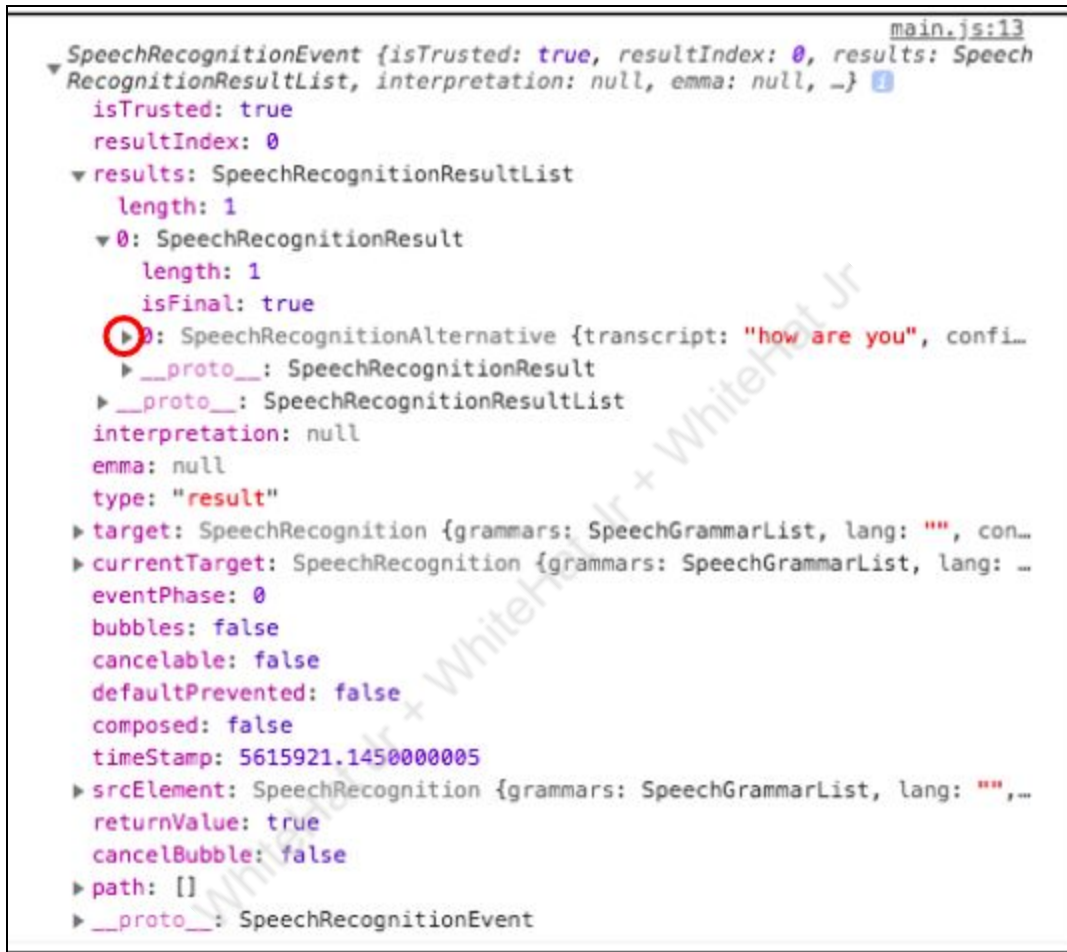
After clicking on 0 which is inside the results:

```
main.js:13
▼ SpeechRecognitionEvent {isTrusted: true, resultIndex: 0, results: Speech
RecognitionResultList, interpretation: null, emma: null, ...} ⓘ
  isTrusted: true
  resultIndex: 0
  ▼ results: SpeechRecognitionResultList
    length: 1
    ▼ 0: SpeechRecognitionResult
      length: 1
      isFinal: true
      ▶ 0: SpeechRecognitionAlternative {transcript: "how are you", confi...
      ▶ __proto__: SpeechRecognitionResult
      ▶ __proto__: SpeechRecognitionResultList
      interpretation: null
      emma: null
      type: "result"
      ▶ target: SpeechRecognition {grammars: SpeechGrammarList, lang: "", con...
      ▶ currentTarget: SpeechRecognition {grammars: SpeechGrammarList, lang: ...
      eventPhase: 0
      bubbles: false
      cancelable: false
      defaultPrevented: false
      composed: false
      timeStamp: 5615921.145000005
      ▶ srcElement: SpeechRecognition {grammars: SpeechGrammarList, lang: "",...
      returnValue: true
      cancelBubble: false
      path: []
      ▶ __proto__: SpeechRecognitionEvent
```

- You will see too much data, but we don't need this data, so click on the sign as per the below image.

7. As we are keeping a track on which section we have clicked till now, we have clicked **event -> results -> 0** till now. In code it will be written as **event.results[0]**.

In the image below we have clicked on the 0 which is inside the 0, so we can say inside the event we have clicked on the results, inside the results we have clicked 0, and inside 0 we have clicked 0 again. In code it will be written as **event.results[0][0]**.



```
main.js:13
▼ SpeechRecognitionEvent {isTrusted: true, resultIndex: 0, results: Speech
RecognitionResultList, interpretation: null, emma: null, ...}
  isTrusted: true
  resultIndex: 0
  results: SpeechRecognitionResultList
    length: 1
    ▼ 0: SpeechRecognitionResult
      length: 1
      isFinal: true
      ► 0: SpeechRecognitionAlternative {transcript: "how are you", confi...
        ► __proto__: SpeechRecognitionResult
        ► __proto__: SpeechRecognitionResultList
      interpretation: null
      emma: null
      type: "result"
      ► target: SpeechRecognition {grammars: SpeechGrammarList, lang: "", con...
      ► currentTarget: SpeechRecognition {grammars: SpeechGrammarList, lang: ...
        eventPhase: 0
        bubbles: false
        cancelable: false
        defaultPrevented: false
        composed: false
        timeStamp: 5615921.1450000005
      ► srcElement: SpeechRecognition {grammars: SpeechGrammarList, lang: "", ...
        returnValue: true
        cancelBubble: false
      ► path: []
      ► __proto__: SpeechRecognitionEvent
```

After clicking on 0 which is inside 0:



So at last we have reached where we wanted to. The marked thing above has the text of our speech, so we want to fetch it.

8. As we are keeping a track on which section we have clicked till now, we have clicked **event** -> **results** -> **0** -> **0** till now. In code it will be written as: **event.results[0][0]**. So to fetch the text form transcript which is inside **0**. The code will be **event.results[0][0].transcript**.
  - Now we know this **-event.results[0][0].transcript** has our speech to text output, so store this inside a variable:

```
var Content = event.results[0][0].transcript;
```

- Now console this variable and see:

```
console.log(Content);
```

- Output on console screen:



how are you

- What we spoke has been successfully generated in text.
- Now we need to update the value of textarea with this value.

```
document.getElementById("textbox").innerHTML = Content;
```

### What's NEXT?

In the next session, we will learn to write a function that will convert text to speech. We will also learn how to configure a webcam.