WhiteHat Jr
Live online Coding for Kids

| Topic | SIMPLIFYING JavaScript - BASICS OF jQuery | |
|---|---|---|
| Class Description | Students will understand the difference between JavaScript and jQuery. Students will also create a simple application to learn about the basics of jQuery. | |
| Class | C176 | |
| Class time | 45 mins | |
| Goal | ● Understand the difference between JavaScript and jQuery.<br>● Learn the basics of jQuery.<br>● Create an online Mad Libs using jQuery. | |
| Resources Required | ● Teacher Resources:<br>○ Visual Studio Code Editor<br>○ laptop with internet connectivity<br>○ smartphone<br>○ earphones with mic<br>○ notebook and pen<br><br>● Student Resources:<br>○ Visual Studio Code Editor<br>○ laptop with internet connectivity<br>○ smartphone<br>○ earphones with mic<br>○ notebook and pen | |
| Class structure | Warm-Up<br>Teacher-led Activity<br>Student-led Activity<br>Wrap-Up | 5 mins<br>15 mins<br>20 mins<br>5 mins |
| WARM-UP SESSION - 5 mins | | |
| CONTEXT | | |

| | | |
|---|---|---|
| ● **Understanding the basics of jQuery.** | | |
| **Teacher Starts Slideshow**<br>**Slide 1 to 3**<br>Refer to speaker notes and follow the instructions on each slide. | | |
| Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today?<br><br>**Following are the WARM-UP session deliverables:**<br>● Greet the student.<br>● Revision of previous class activities. | | **ESR**: Hi, thanks!<br>Yes I am excited about it!<br><br>Click on the slide show tab and present the slides |
| **WARM-UP QUIZ**<br>Click on In-Class Quiz | | |
| **Continue WARM-UP Session**<br>**Slide 4 to 19** | | |
| **Following are the session deliverables:**<br>● Appreciate the student.<br>● Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students. | | |
| **Class Steps** | **Teacher Action** | **Student Action** |
| **Step 1:**<br>**Warm-Up**<br>**(5 mins)** | Hi, how are you?<br><br>Great! | **ESR:** I am good! |
| | Augmented reality has been fun till now. We have been using markers to create Augmented reality scenes. | |

| | | |
|---|---|---|
| | Do you remember why we use markers for AR?<br><br>Great!<br><br>When we run the Augmented reality application, the camera opens and we need something to identify what is the starting point of that Augmented reality scene.<br><br>Markers are a way to set the origin of the Augmented reality world space within the camera view.<br><br>In upcoming classes, we will see how to create **web AR applications without markers**.<br><br>So how do you think we can render any object without markers?<br><br>Well, we need the starting point inside the camera view to render objects and if you don't have the markers there has to be something else, which is going to tell us the starting point of an augmented world space.<br><br>There are multiple ways to do this but one way could be using your current location. | **ESR**: Markers help us to render objects when we scan them.<br><br>**ESR**: Varied. |

| | | |
|---|---|---|
| | It's easy to find out your current location nowadays with advanced technologies like GPS (**Global Positioning System**).<br><br>We will see how this technology can help us to find the location, then how we can combine it with Augmented reality.<br><br>Before we can start learning and implementing these functionalities we need to learn a way to **simplify the JavaScript code writing** because implementing these technologies in JavaScript is going to increase load on the application, making them very difficult to run.<br><br>This can be done with another JavaScript library known as **jQuery**, which is specially designed to **"write less, do more"**.<br><br>As programmers, we should be adaptable to the new technologies for languages which can help us make better applications.<br><br>In today's class we are going to start with the basics of **jQuery**.<br><br>We will make a simple **Mad Libs** application using **jQuery**.<br><br>Have you ever played Mad Libs? | **ESR**: Yes/No. |

| | | |
|---|---|---|
| | Mad Libs are short and silly stories based on your words.<br><br>In Mad Libs, there will be stories with some blanks to fill in. It is played by filling up the blanks for a given story.<br><br>We can either provide a word bank, i.e. a list of words to choose from while filling the blanks or we can let the user use any word they'd like.<br><br>Since we want to do some scoring in our version of mad libs, we will give the user a list of words to choose from while filling the blanks!<br><br>So let's get started then. | |

**Teacher Ends Slideshow**

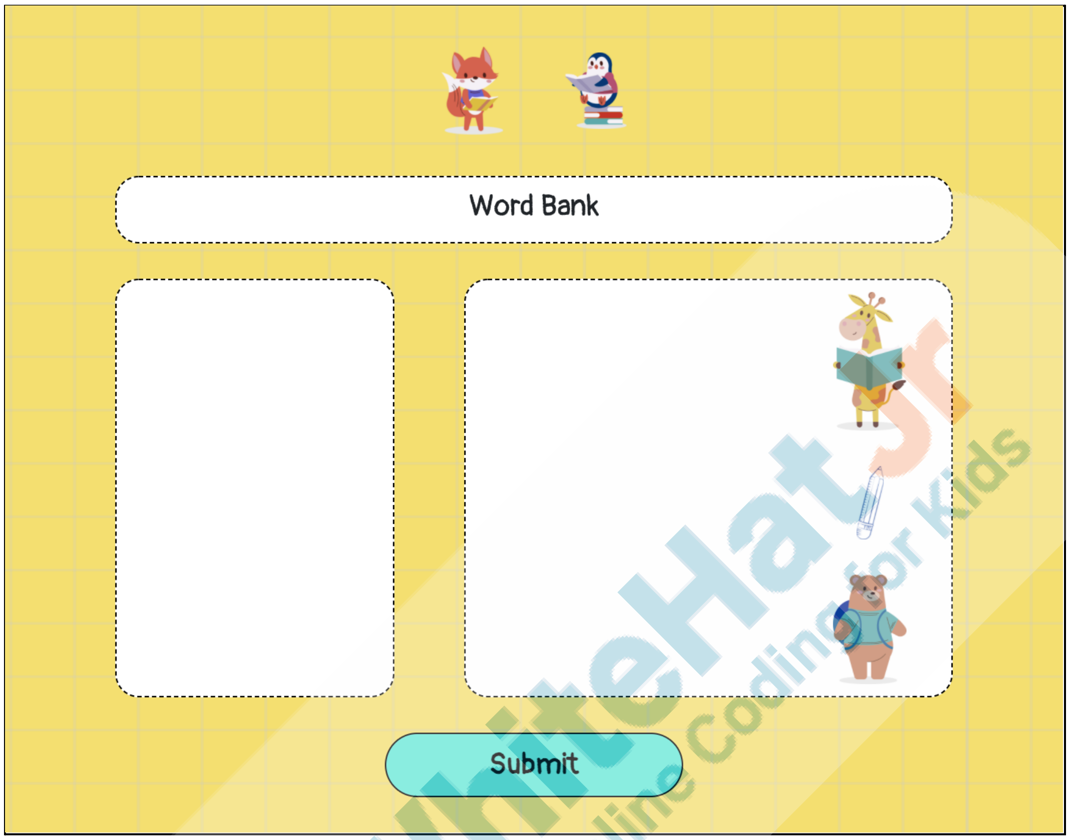
**TEACHER-LED ACTIVITY - 15 mins**

**Teacher Initiates Screen Share**

**CHALLENGE**

- **Understand the difference between JavaScript and jQuery.**
- **Create Mad Libs using jQuery.**

| | | |
|---|---|---|
| **Step 2: Teacher-led Activity (15 mins)** | *<The teacher clones the activity from the Teacher Activity 1>.*<br><br>***[Teacher Activity 1]***<br><br>What do you think is the first thing that we need to start making an online Mad Libs application?<br><br>To start with, we need to design an HTML page where we can show:<br>● List of words for hints to complete the story.<br>● Some input boxes users can type to fill in the blanks.<br>● The stories' text with some fill in the blanks.<br><br>*<The teacher runs the code and shows the output>.* | <br><br><br><br>**ESR**: Varied. |

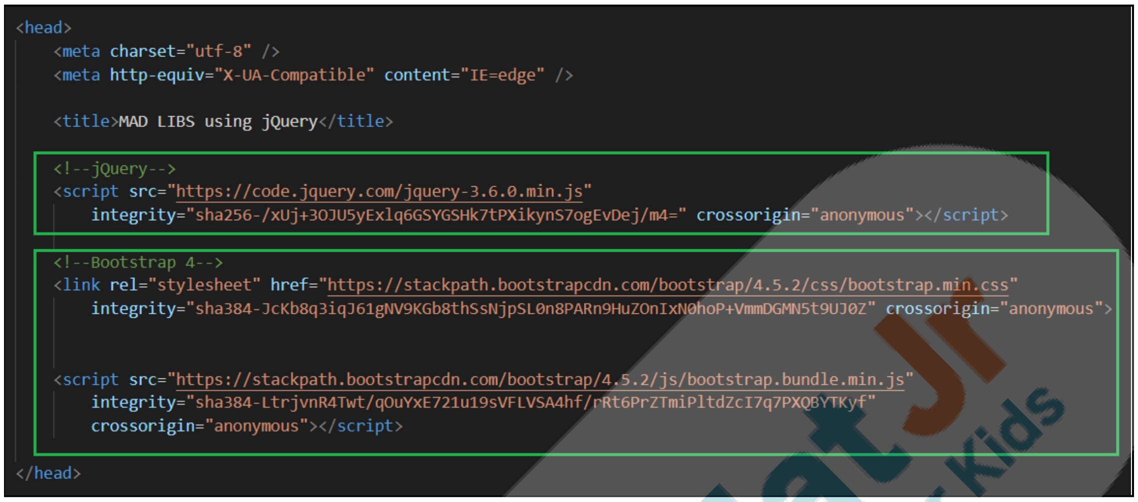


Let’s take a walkthrough of the code and understand how this template is made.

We have used Bootstrap to design this simple template for our Mad Libs game.

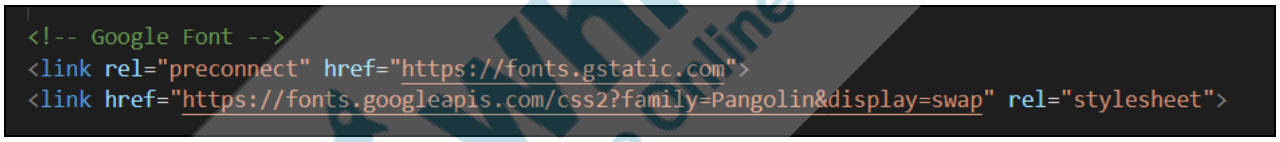
The library link is added in the <head> tag.

***Note**: The below code is already a part of the boilerplate.*



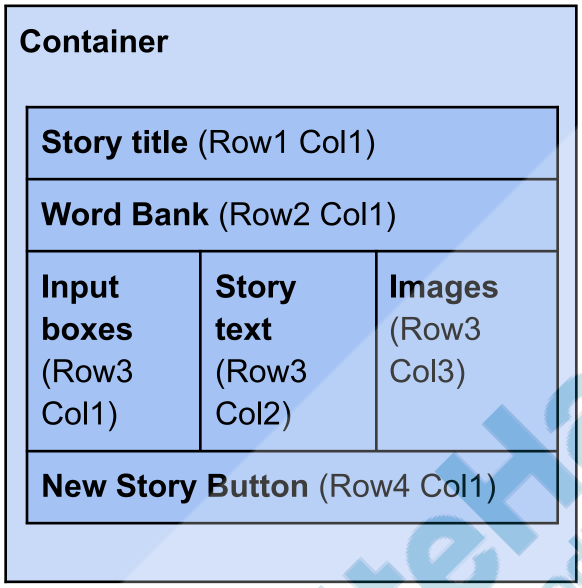
```
<head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <title>MAD LIBS using jQuery</title>

    <!--jQuery-->
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"
        integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4=" crossorigin="anonymous"></script>

    <!--Bootstrap 4-->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
        integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z" crossorigin="anonymous">


    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.bundle.min.js"
        integrity="sha384-LtrjvnR4Twt/qOuYxE721u19sVFLVSA4hf/rRt6PrZTmiPltdZcI7q7PXQBYTKyf"
        crossorigin="anonymous"></script>
</head>
```

Just as we have Bootstrap and jQuery added into our boilerplate code, we also have added Google Fonts since we are not using the default fonts.

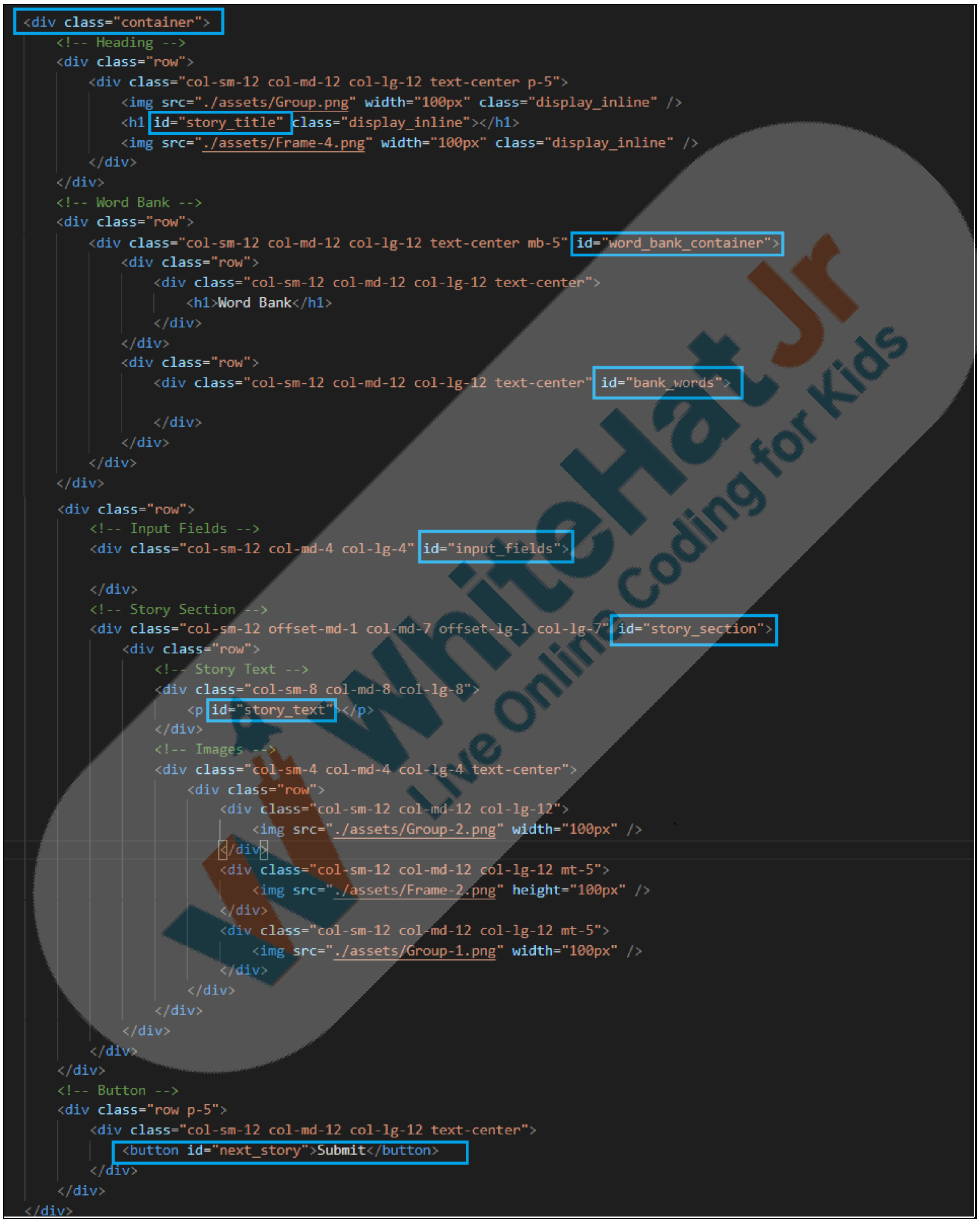


```
<!-- Google Font -->
<link rel="preconnect" href="https://fonts.gstatic.com">
<link href="https://fonts.googleapis.com/css2?family=Pangolin&display=swap" rel="stylesheet">
```

The HTML elements are added in the following DOM structure:

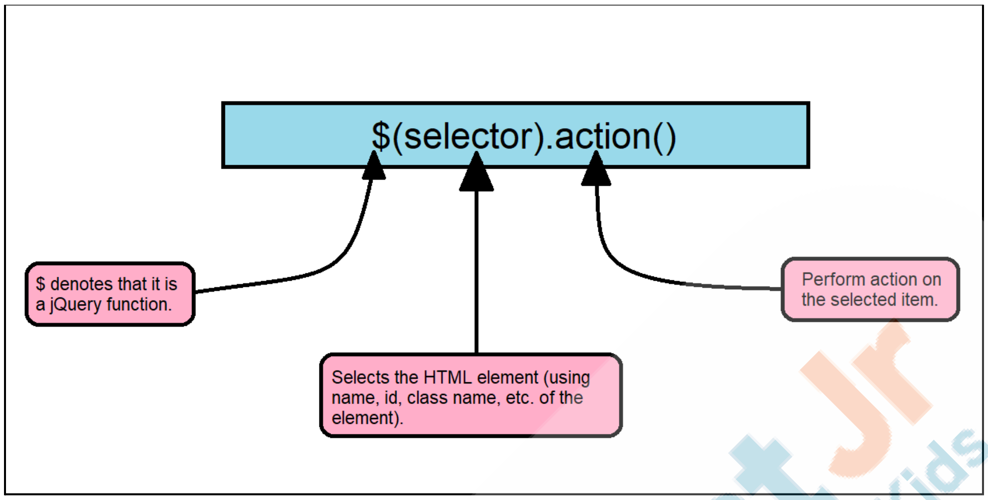


***Note**: The below code is already a part of the boilerplate.*



```
<div class="container">
    <!-- Heading -->
    <div class="row">
        <div class="col-sm-12 col-md-12 col-lg-12 text-center p-5">
            <img src="./assets/Group.png" width="100px" class="display_inline" />
            <h1 id="story_title" class="display_inline"></h1>
            <img src="./assets/Frame-4.png" width="100px" class="display_inline" />
        </div>
    </div>
    <!-- Word Bank -->
    <div class="row">
        <div class="col-sm-12 col-md-12 col-lg-12 text-center mb-5" id="word_bank_container">
            <div class="row">
                <div class="col-sm-12 col-md-12 col-lg-12 text-center">
                    <h1>Word Bank</h1>
                </div>
            </div>
            <div class="row">
                <div class="col-sm-12 col-md-12 col-lg-12 text-center" id="bank_words">

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

    <div class="row">
        <!-- Input Fields -->
        <div class="col-sm-12 col-md-4 col-lg-4" id="input_fields">

        </div>
        <!-- Story Section -->
        <div class="col-sm-12 offset-md-1 col-md-7 offset-lg-1 col-lg-7" id="story_section">
            <div class="row">
                <!-- Story Text -->
                <div class="col-sm-8 col-md-8 col-lg-8">
                    <p id="story_text"></p>
                </div>
                <!-- Images -->
                <div class="col-sm-4 col-md-4 col-lg-4 text-center">
                    <div class="row">
                        <div class="col-sm-12 col-md-12 col-lg-12">
                            <img src="./assets/Group-2.png" width="100px" />
                        </div>
                        <div class="col-sm-12 col-md-12 col-lg-12 mt-5">
                            <img src="./assets/Frame-2.png" height="100px" />
                        </div>
                        <div class="col-sm-12 col-md-12 col-lg-12 mt-5">
                            <img src="./assets/Group-1.png" width="100px" />
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <!-- Button -->
    <div class="row p-5">
        <div class="col-sm-12 col-md-12 col-lg-12 text-center">
            <button id="next_story">Submit</button>
        </div>
    </div>
</div>
```

| | | |
|---|---|---|
| ***Note**: And some styling has been added to these elements using id and class names. Check out **index.css** file of the boilerplate code.* | | |
| | Now we are going to add functions to display story text and input boxes using jQuery.<br><br>Let's first understand how instructions are written in jQuery.<br><br>For this first we need to include the jQuery library.<br><br>The basic syntax of the jQuery is:<br><br>**$(selector).action()**<br><br>**$**: The dollar symbol denotes the beginning of a jQuery function.<br><br>**selector**: It is the id, tag name, class name of the HTML element that we want to select.<br><br>**action**: It is a jQuery action that can be performed on the selected element, like, setting an attribute or hiding an element.<br><br>*The teacher opens the reference document and discusses the syntax.*<br><br>**jQuery Syntax Reference Document** | |

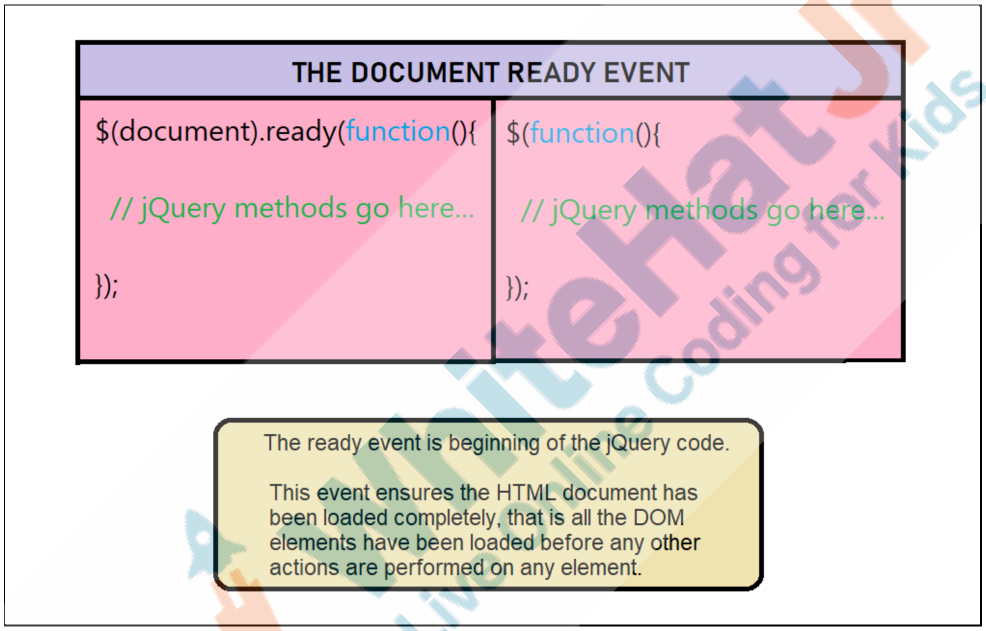

$(selector).action()
$ denotes that it is a jQuery function.
Selects the HTML element (using name, id, class name, etc. of the element).
Perform action on the selected item.

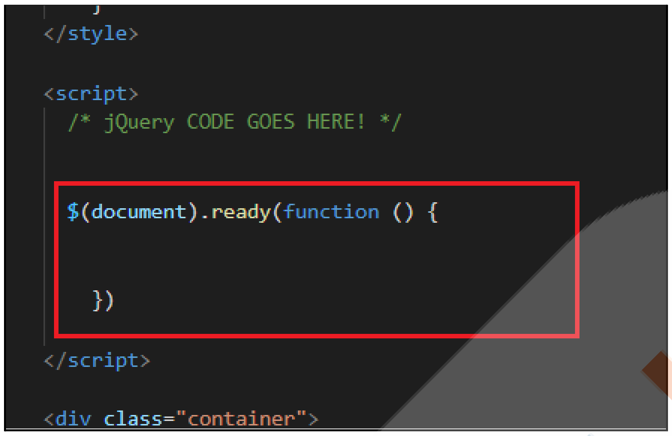
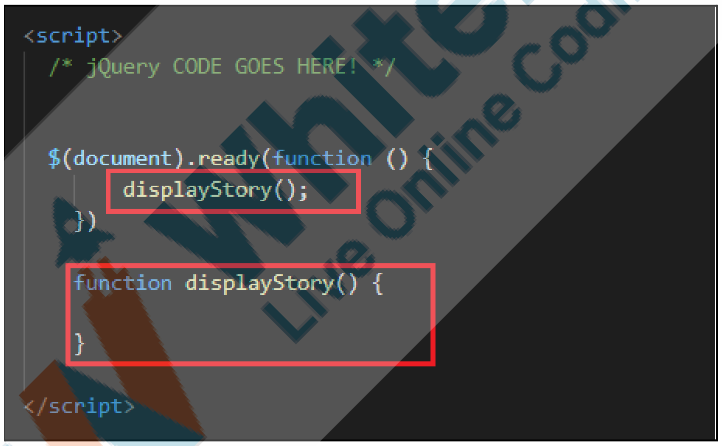
The **ready event** is the beginning of the jQuery actions that can be performed on the elements.

This event ensures that all elements on the HTML page (or HTML document) have been loaded properly.

There are two ways to write this event:

```
$(document).ready(function(){

 //actions once all the DOM
elements      are loaded…

});
                 OR
$(function(){

  //actions once all the DOM
elements      are loaded…
```

|  |  |  |
|---|---|---|
|  | `});`<br><br>*The teacher opens the reference document and discusses the syntax.*<br><br>[jQuery Syntax Reference Document]() |  |
| <br>**THE DOCUMENT READY EVENT**<br>`$(document).ready(function(){`<br>`  // jQuery methods go here...`<br>`});`<br>`$(function(){`<br>`  // jQuery methods go here...`<br>`});`<br>The ready event is beginning of the jQuery code.<br>This event ensures the HTML document has been loaded completely, that is all the DOM elements have been loaded before any other actions are performed on any element. |  |  |
|  | Let's write this method in our code.<br><br>Since jQuery is just the simplified version of JavaScript, we will add the jQuery code, also under the &lt;script&gt; tag.<br><br>*&lt;The teacher adds the document ready event.&gt;* |  |

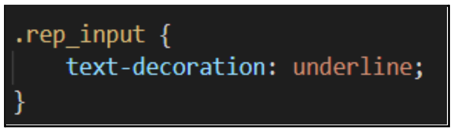
| | | |
|---|---|---|
| `</style>` `<script>` `/* jQuery CODE GOES HERE! */` `$(document).ready(function () {` `})` `</script>` `<div class="container">` | | |
| | Now we are going to write a function **displayStory()**, which will show the text content of the story with a few fill in the blanks in the story content. | |
| `<script>` `/* jQuery CODE GOES HERE! */` `$(document).ready(function () {` `displayStory();` `})` `function displayStory() {` `}` `</script>` | | |
| | To display the story content on the HTML page, let's have some information about the story.<br><br>Can you tell me what information we will need to show the story with a few fill in the blanks? | **ESR:**<br>● We can have the title of the story and |

```
    }
</style>

<script>
  /* jQuery CODE GOES HERE! */

  $(document).ready(function () {


    })
</script>

<div class="container">
```

```
<script>
  /* jQuery CODE GOES HERE! */

  $(document).ready(function () {
      displayStory();
  })

  function displayStory() {

  }
</script>
```

| | | |
|---|---|---|
| | Yes! Great!<br><br>Let's keep this data in a variable array of JSON objects.<br><br>*<The teacher explains story data in the boilerplate.>*<br><br>var **stories**=[<br>{<br>"**inputs**": number,<br>"**title**": "title of the story",<br>"**story**": `story content`,<br>"**words**":array of words<br>}<br>]<br><br>● "**inputs**": will be equal to the number of fill in the blanks in the the story<br>● "**title**": title of the story<br>● "**story**": content of the story<br>● "**words**": array of words for hint for the answers<br><br>***Note**: Make sure the story text is properly written inside ` `.*<br><br>To add dashes (____) in the story content, we are going to use the | the content of the story.<br><br>● For the blanks we can use dashes. |

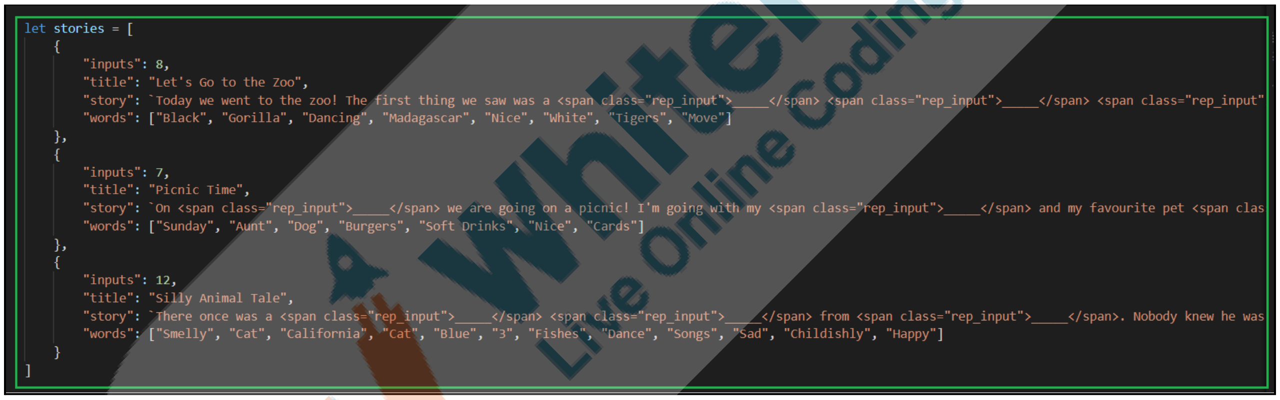
| | | |
|---|---|---|
| | **<span></span>** tag.<br><br><span> tags are similar to <div> tags, but the only difference is that <span> tags are inline tags while <div> tags are block tags.<br><br>This means that if we use two <div> tags, it will be displayed in two different lines, but two <span> tags can be displayed side by side.<br><br>In our case, we are using the <span> tag in between the text.<br><br>With this, we can provide a class or an id to our dashes (______), to refer to these in between items later without affecting all the other text around it.<br><br>All dashes within the story will be inside the <span> tag:<br><br>**<span class=”rep_input”>____</span>**<br><br>**rep_input**:<br>This is a class name we are providing to our dashes. With this, we can -<br><br>● Style our text inside span tag in a way that it is underlined through CSS. | |



```
.rep_input {
    text-decoration: underline;
}
```

- And later on we can refer to these span tags using the class name and update the contents of it. This means that when the user enters a word, we can fill the dashes (_____) with the word the user has entered. The rest of the text around this tag will remain the same.

*Note: The below code is already a part of the boilerplate.*



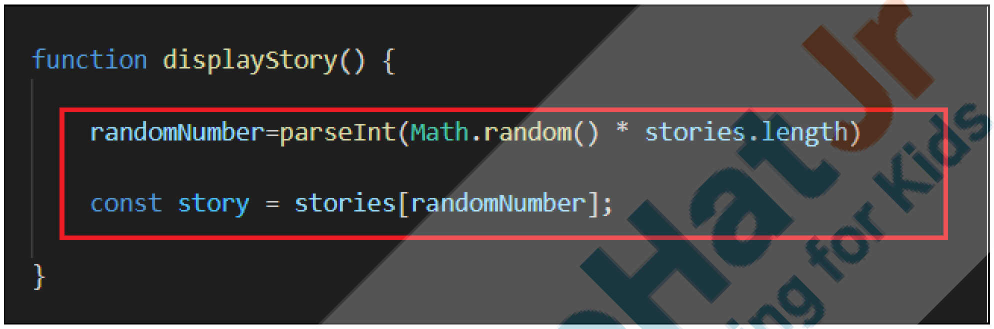
```
let stories = [
    {
        "inputs": 8,
        "title": "Let's Go to the Zoo",
        "story": `Today we went to the zoo! The first thing we saw was a <span class="rep_input">_____</span> <span class="rep_input">_____</span> <span class="rep_input"
        "words": ["Black", "Gorilla", "Dancing", "Madagascar", "Nice", "White", "Tigers", "Move"]
    },
    {
        "inputs": 7,
        "title": "Picnic Time",
        "story": `On <span class="rep_input">_____</span> we are going on a picnic! I'm going with my <span class="rep_input">_____</span> and my favourite pet <span clas
        "words": ["Sunday", "Aunt", "Dog", "Burgers", "Soft Drinks", "Nice", "Cards"]
    },
    {
        "inputs": 12,
        "title": "Silly Animal Tale",
        "story": `There once was a <span class="rep_input">_____</span> <span class="rep_input">_____</span> from <span class="rep_input">_____</span>. Nobody knew he was
        "words": ["Smelly", "Cat", "California", "Cat", "Blue", "3", "Fishes", "Dance", "Songs", "Sad", "Childishly", "Happy"]
    }
]
```

Now let's continue writing the **displayStory()** function:

- First we will randomly pick up the story from the array list of **stories**.

- Then **stories.length** can give us the count of stories present in the array.

| | | |
|---|---|---|
| | • We can use **Math.random()** to find the random number between (**0 to stories.length**).<br>• Use **parseInt()** to remove decimal points. | |



```
function displayStory() {
    randomNumber=parseInt(Math.random() * stories.length)

    const story = stories[randomNumber];
}
```

| | | |
|---|---|---|
| | Now we are going to first set the title of the story to show on the HTML page.<br><br>Do you remember which method we should use to set the elements/attributes of the HTML page?<br><br>But today, we will do this using jQuery.<br><br>Let's first understand the syntax to **set the content using jQuery**.<br><br>There are four methods in jQuery to set the content.<br><br>*The teacher opens the reference doc and discusses the syntax.*<br><br>[jQuery Syntax Reference Document] | **ESR:** We should use the **setAttribute()** method. |

| SET DOM CONTENT | | |
|---|---|---|
| | **JavaScript** | **jQuery** |
| How? | The **setAttribute()** method adds the specified attribute to an element, and gives it the specified value. If the specified attribute already exists, only the value is set/changed. | **text()** - Sets the text content of selected elements.<br><br>**html()** - Sets the content of selected elements (including HTML markup).<br><br>**val()** - Sets the value of form fields.<br><br>**attr()** method is used to set attribute values. |
| Syntax | element.setAttribute(attributename, attributevalue) | $(selector).text("some text")<br><br>$(selector).attr(attributename, attributevalue); |
| Example | var el = document.getElementById("ref");<br><br>el.setAttribute("href","https://www.whitehatjr.com/"); | $(selector).text("Hi There!")<br><br>$(this).attr("href","https://www.whitehatjr.com/"); |

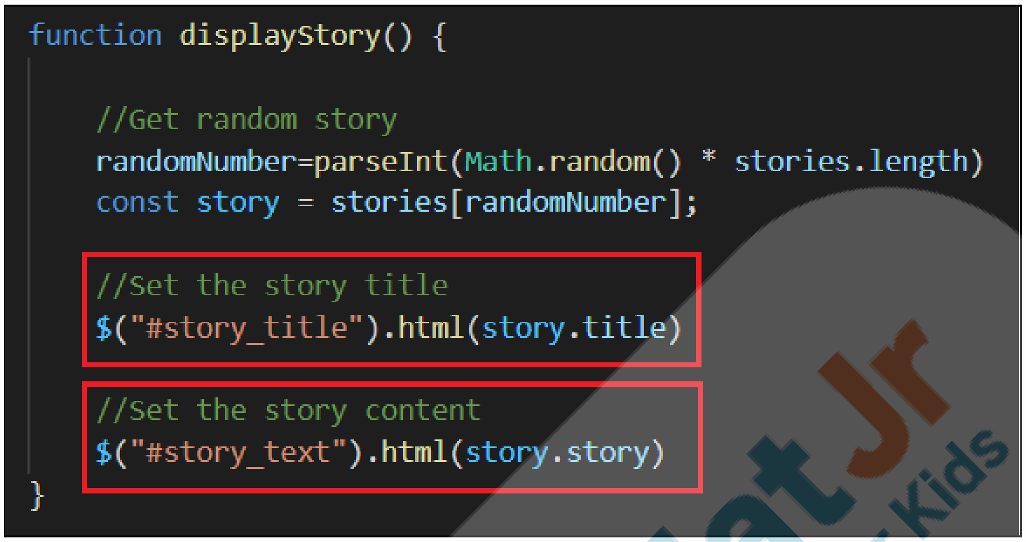
Now we are going to use the **html()** method to set the story title and the story content and see the output:

- **Select the div element using id selector**:

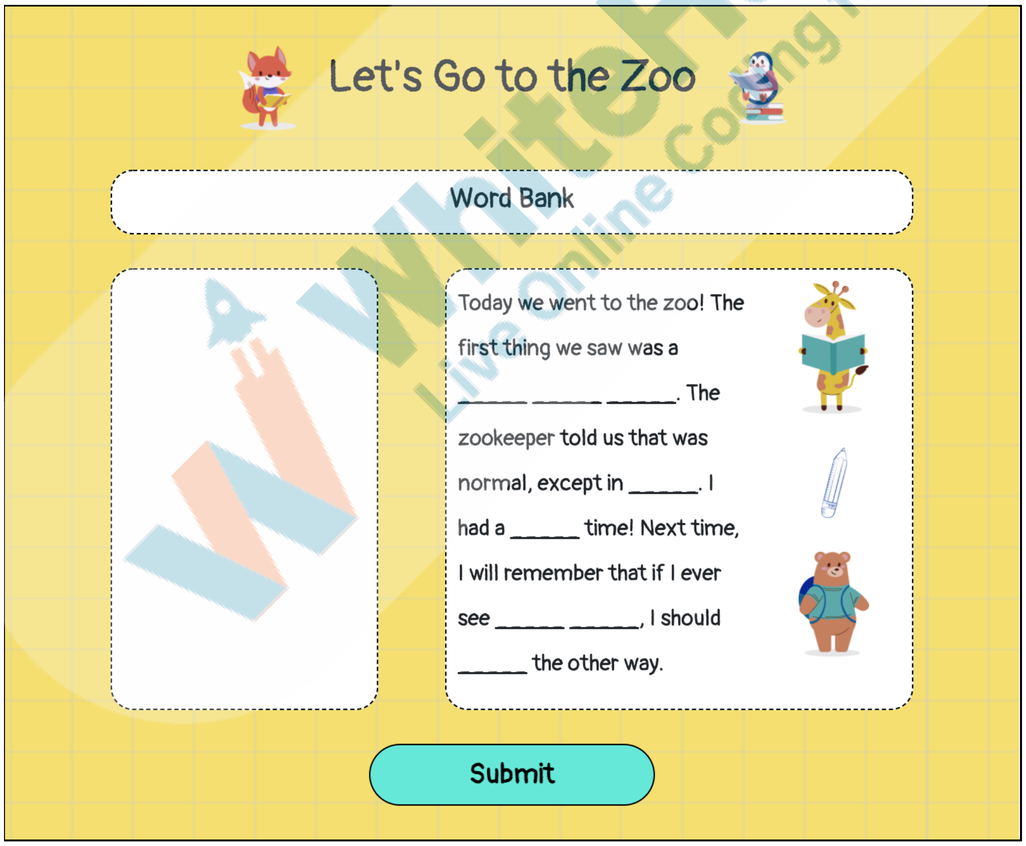
  $("#story_title)

- **Use .html() method and pass the value**:

  $("#story_title).html(story.title)



```
function displayStory() {

    //Get random story
    randomNumber=parseInt(Math.random() * stories.length)
    const story = stories[randomNumber];

    //Set the story title
    $("#story_title").html(story.title)

    //Set the story content
    $("#story_text").html(story.story)
}
```

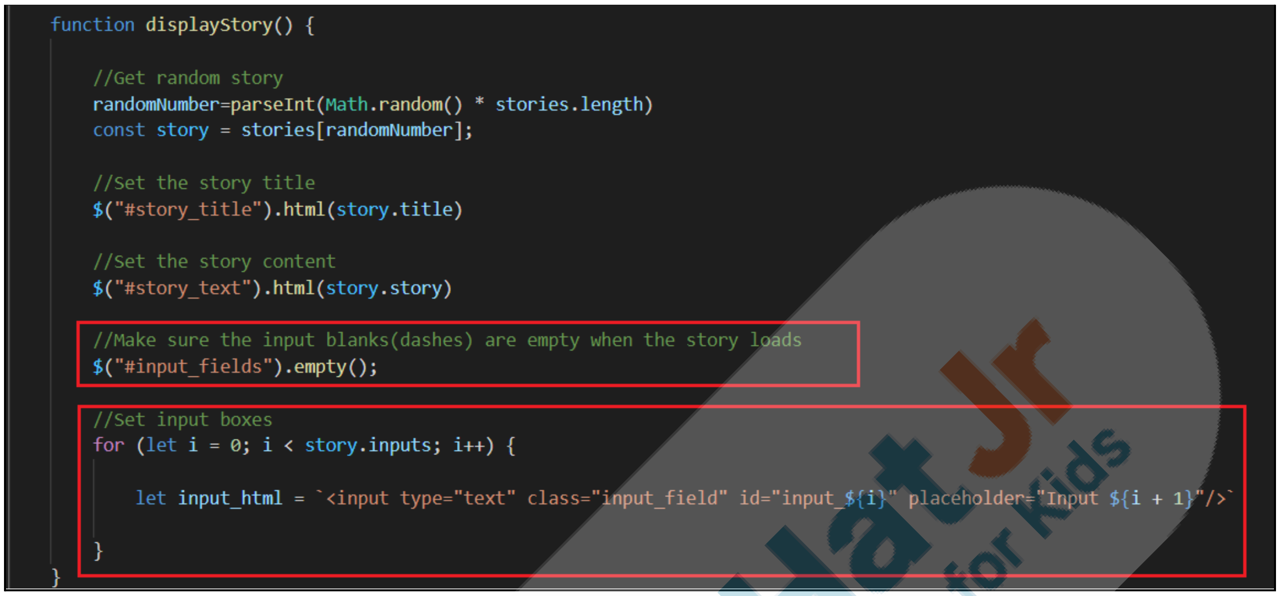


Let's Go to the Zoo

Word Bank

Today we went to the zoo! The first thing we saw was a ______ ______ ______. The zookeeper told us that was normal, except in ______. I had a ______ time! Next time, I will remember that if I ever see ______ ______, I should ______ the other way.

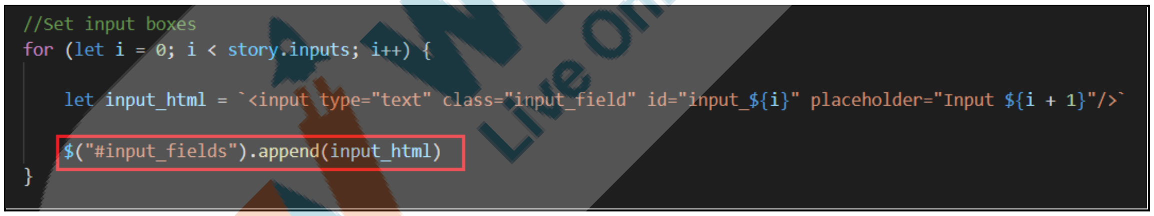
| | | |
|---|---|---|
| | Now we are going to **add input boxes for all the blanks** (dashes) on the left side of the story content.<br><br>For example, if we have 5 blanks to fill in the story, we will need 5 boxes on the left side to type words.<br><br>Can you tell me which HTML tag we should use to set the input fields where users can type the words?<br>Amazing!<br><br>Remember we have stored the number of inputs in the **stories** array?<br><br>We are going to loop through all the blanks and set the <input> tag.<br><br>For the <input> tag, set attribute:<br>● **type**: type of the input field.<br>● **class**: class of the input field for styling the input tag. This will also help us get all the input tags to fetch values later. (We'll see how.)<br>● **id**: id of the input field to uniquely identify all the input fields.<br>● **placeholder**: small text in the box which tells what is expected to be filled. | **ESR:** The <input> tag is used to set the input field where users can enter data.<br><br>**ESR:** Yes. |



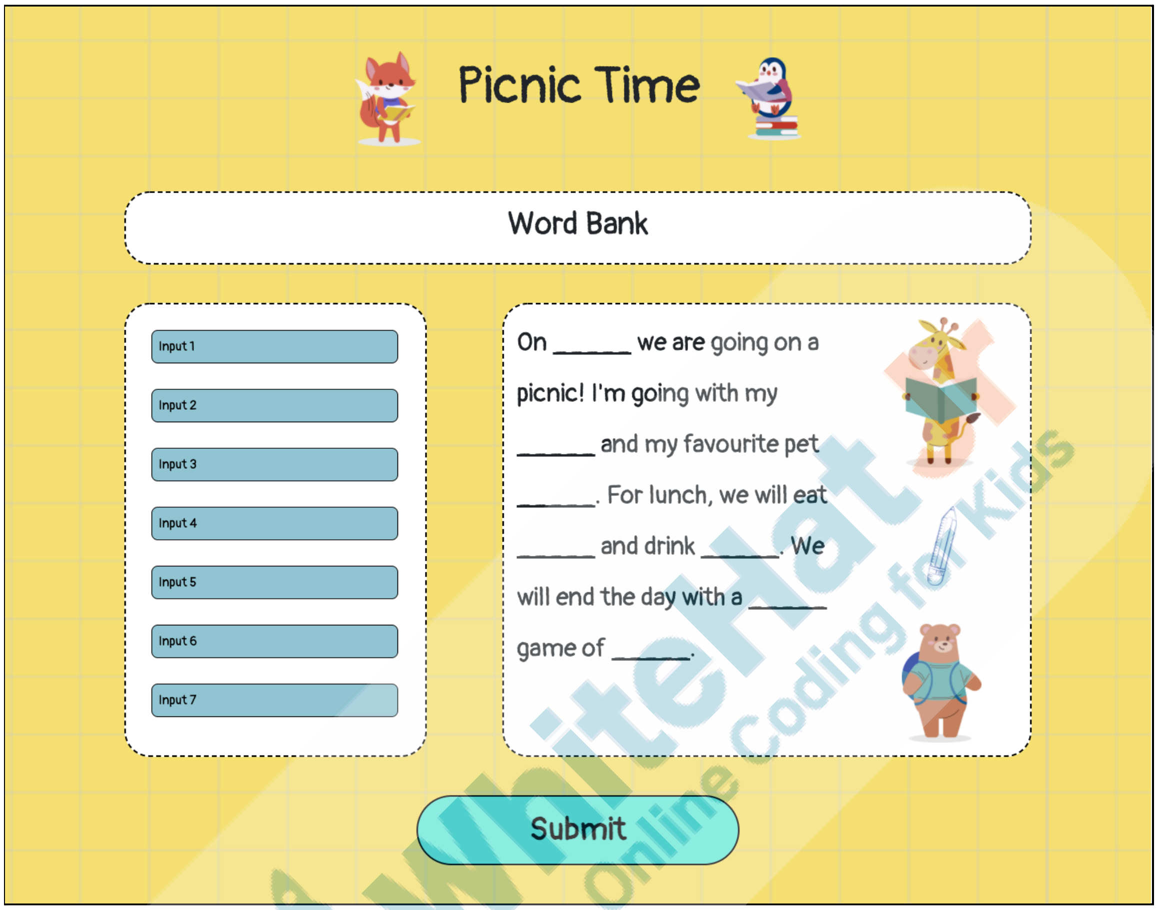
```
function displayStory() {

    //Get random story
    randomNumber=parseInt(Math.random() * stories.length)
    const story = stories[randomNumber];

    //Set the story title
    $("#story_title").html(story.title)

    //Set the story content
    $("#story_text").html(story.story)

    //Make sure the input blanks(dashes) are empty when the story loads
    $("#input_fields").empty();

    //Set input boxes
    for (let i = 0; i < story.inputs; i++) {

        let input_html = `<input type="text" class="input_field" id="input_${i}" placeholder="Input ${i + 1}"/>`

    }
}
```

| | | |
|---|---|---|
| | Once we set the HTML input field, we will need to append all the boxes one after the other.<br><br>In jQuery we also have the **append()** method.<br><br>We will use it to append the input field one after the other.<br><br>*The teacher opens the reference document and discusses the append() method syntax.*<br><br>**[jQuery Syntax Reference Document]** | |

| APPEND ELEMENT | | |
|---|---|---|
| | **JavaScript** | **jQuery** |
| How? | The **appendChild()** method appends element node to another element node. | **append()** - Inserts content at the end of the selected elements.<br><br>**prepend()** - Inserts content at the beginning of the selected elements.<br><br>**after()** - Inserts content after the selected elements.<br><br>**before()** - Inserts content before the selected elements. |
| Syntax | elementNode1.appendChild(elementNode2) | $(selector).append(element) |
| Example | var sceneEl = document.createElement("a-scene");<br>var el = document.createElement("a-entity");<br>sceneEl.appendChild(el); | $("p").append("some text appended."); |



```
//Set input boxes
for (let i = 0; i < story.inputs; i++) {

    let input_html = `<input type="text" class="input_field" id="input_${i}" placeholder="Input ${i + 1}"/>`

    $("#input_fields").append(input_html)
}
```


Picnic Time

Word Bank

Input 1
Input 2
Input 3
Input 4
Input 5
Input 6
Input 7

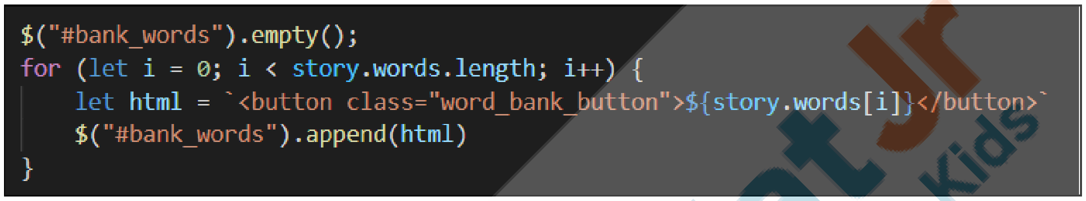
On _____ we are going on a picnic! I'm going with my _____ and my favourite pet _____. For lunch, we will eat _____ and drink _____. We will end the day with a _____ game of _____.

***Note:*** *We already had added styling in our boilerplate code for the class* ***input_field*** *that we have used for our input fields; therefore the input fields are styled by default on adding this code.*

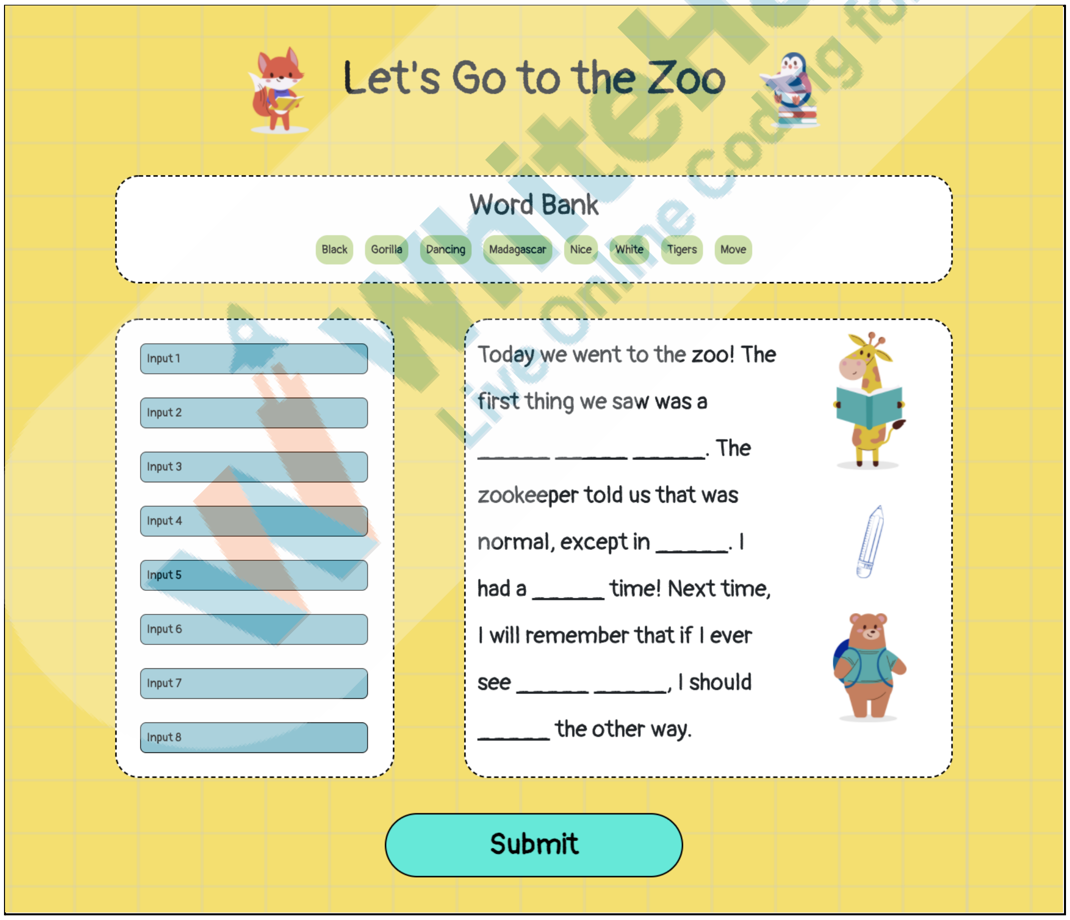
| | | |
|---|---|---|
| | Similar to how we added input fields, we will add words that we have in our stories one by one in our word bank's box.<br><br>Again, we have the id **#bank_words** on the container inside which we want to add these words. | |

So we can simply iterate over all the words and use **<button>** elements with class **word_bank_button**.

This is because we already had styling added for class **word_bank_button** in our boilerplate code.



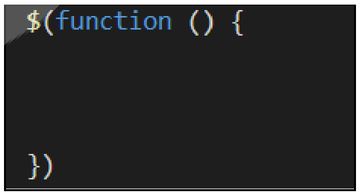
```
$("#bank_words").empty();
for (let i = 0; i < story.words.length; i++) {
    let html = `<button class="word_bank_button">${story.words[i]}</button>`
    $("#bank_words").append(html)
}
```

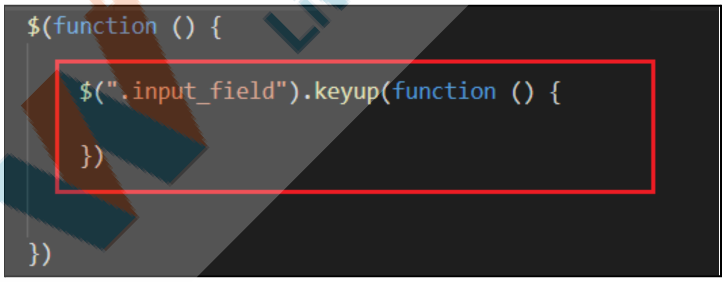


Today we went to the zoo! The first thing we saw was a ______ ______ ______. The zookeeper told us that was normal, except in ______. I had a ______ time! Next time, I will remember that if I ever see ______ ______, I should ______ the other way.

| | | |
|---|---|---|
| | Now we have story text and the input boxes to fill the words in the blanks. We also have our word bank in place, for users to choose the words from for the blanks.<br><br>But right now when we type in the boxes the blanks do not fill up.<br><br>Now you will write a jQuery function that will help the user fill in the blanks as soon as the user types in the boxes. This will be happening in real-time!<br><br>That will be cool, isn't it?<br><br>Are you excited? | <br><br><br><br><br><br><br><br><br><br>**ESR**: Yes!<br><br>**ESR**: Yes! |
| **Teacher Stops Screen Share** | | |
| | Now it's your turn. Please share your screen with me. | |
| **Teacher Starts Slideshow**<br>**Slide 20 to 21**<br>Refer to speaker notes and follow the instructions on each slide. | | |
| We have one more class challenge for you.<br>Can you solve it?<br><br>Let's try. I will guide you through it. | | |
| **Teacher Ends Slideshow** | | |

| STUDENT-LED ACTIVITY - 20 mins | | |
|---|---|---|
| • Ask the student to press the ESC key to come back to the panel.<br>• Guide the student to start screen share.<br>• Teacher gets into fullscreen. | | |
| **ACTIVITY**<br>• Write a jQuery function to fill the blanks in Mad Libs' story when the user types the words in the input boxes. | | |
| **Step 3: Student-Led Activity (20 mins)** | *The teacher guides the student to clone the code from Student Activity 1.*<br>***[Student Activity 1]***<br><br>***Note**: The student will continue to add new functionality after teacher activity.* | |
| | If we want to update the blanks in the story as soon as the user types something, do you know what we should do? | **ESR:**<br>We can add a listener to the input field to update the story as soon as it changes. |
| | Great!<br><br>Now there can be many different types of listeners that we can have in jQuery.<br><br>We can create listeners for -<br>• Click Events<br>• Change Events<br>• Hover Events<br>• Keypress Events<br>• Scroll Events<br><br>And many more! | |

| | | |
|---|---|---|
| | What event can we use for listening to input field change?<br><br>Great!<br><br>And for the button that we have to go to next story, what event can listen to it?<br><br>Awesome! | **ESR:**<br>We can use the Keypress event on the input field.<br><br>**ESR:**<br>We can use the click event on the button. |
| | Now to add event listeners in jQuery, we use the following syntax we discussed above.<br><br>Syntax:<br>`$(function(){`<br>`    // event listeners`<br>`})`<br><br>We can call it the **dollar function.**<br><br>Inside these dollar functions, we can have as many event listeners on as many elements of HTML as we like.<br><br><br>`$(function () {`<br>`})` | |

|  |  |  |
|---|---|---|
|  | Let's start by adding the **keyup event on the input fields**.<br><br>When we created the input fields, we assigned a class **input_field** to all the input fields.<br><br>Now since we want to listen to a **keyup** event on all the input fields that we have, we can use this class. Therefore:<br><br>● We will then select the input fields using class name:<br><br>$(".input_field")<br><br>● Then we will add the keyup event:<br><br>$(".input_field").keyup(function(){<br><br>} |  |
| <br>`$(function () {`<br>`    $(".input_field").keyup(function () {`<br>`    })`<br>`})` |  |  |
|  | Okay, so the event function is triggered, but for which input field!<br><br>If we go back to where we created the input fields, we added an **id** to all the |  |

| | | |
|---|---|---|
| | input fields to uniquely identify each of them.<br><br>The id was in the format of **index_{i}** where **i** was the input field number that we are creating.<br><br>Now if we think about it, if we update the first input field, then the first blank should be updated.<br><br>Similarly, the second input field should update the second blank, so on and so forth.<br><br>Therefore, if we find out which input number it is by getting its ID, we will know which blank we need to update!<br><br>For finding the input number, we will:<br><br>● **Select the element:**<br><br>When an event is triggered, we get the element on which the event happened in a special keyword "**this**" in JavaScript.<br><br>In our case, if the 3rd input field triggered the event, then the variable **"this"** will contain the third input field's HTML.<br><br>We will use **this** in jQuery also to select the element: | |

$(this)

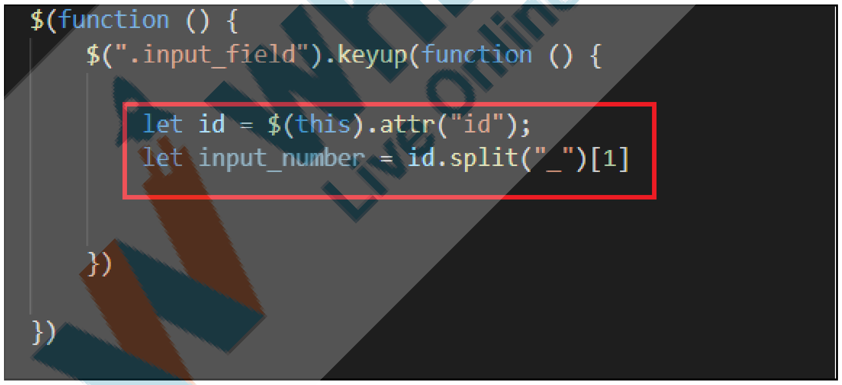
- Get the **id** of that input field using the **attr()** function of jQuery:

  $(this).attr("id")

*The teacher opens the reference document and discusses the attr() syntax.*

[jQuery Syntax Reference Document]

Once we have the ID, we split the id from an underscore ( _ ) and take the first element. This gives us the input number.



```
$(function () {
    $(".input_field").keyup(function () {

        let id = $(this).attr("id");
        let input_number = id.split("_")[1]

    })
})
```

| GET DOM CONTENT | | |
|---|---|---|
| | **JavaScript** | **jQuery** |
| How? | The **getAttribute()** method returns the value of the attribute with the specified name, of an element. | **text()** - Returns the text content of selected elements.<br><br>**html()** - Returns the content of selected elements (including HTML markup).<br><br>**val()** - Returns the value of form fields<br><br>**attr()** method is used to get attribute values. |
| Syntax | element.getAttribute(attributename) | $(selector).text()<br><br>$(selector).attr(attributename); |
| Example | var el = document.getElementById("ref");<br><br>el.getAttribute("href"); | var someText=$("#test").text()<br><br>var id = $(this).attr("href"); |

Now all we had to do was to update the corresponding blank in our story based on the input number.

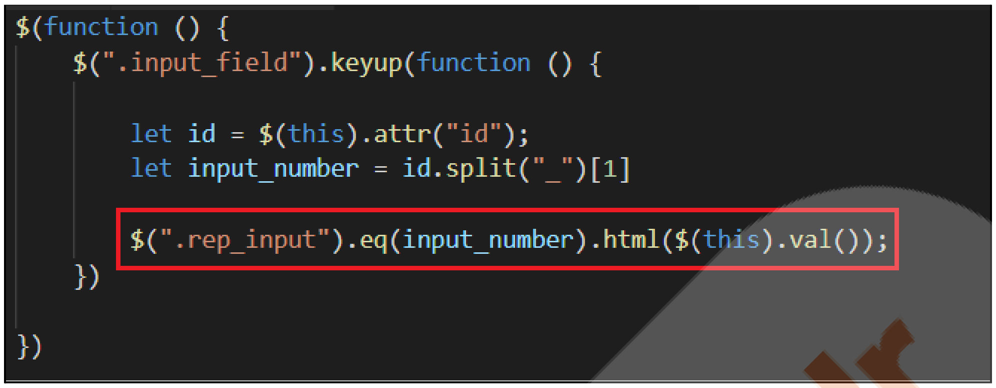
For this, we will:

- **Select the <span> tags:**
  Now since we assigned the **"rep_input"** class to all <span> tags in our story, we can use the class name to select all the <span> tags using jQuery.

  $(".rep_input")

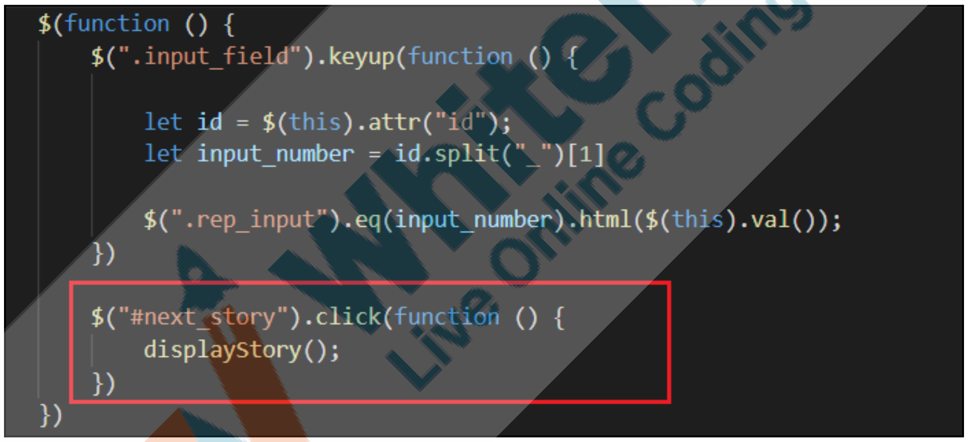
- **Select the specific <span> tag:**
  Once we find all the <span> tags, we can find the specific <span>

| | | |
|---|---|---|
| | tag with index of **input_number**. For this, we have a special function **eq()** in jQuery.<br><br>**Syntax:** $(selector).**eq**(index)<br><br>Using this function we can get the &lt;span&gt; tag on a specific index.<br><br>`$(".rep_input").eq(input_number)`<br><br>● **Add the value to fill in the blanks:**<br>Once we have the &lt;span&gt; tag selected, we will use the **.html()** method on it to update it's HTML.<br><br>`$(".rep_input").eq(input_number).html()`<br><br>The HTML should be updated with the value of the input field, and we have the HTML of the input field in **"this"** variable, therefore we can get the value of our input with **$(this).val()**.<br><br>`$(".rep_input").eq(input_number).html($(this).val())` | |



```
$(function () {
    $(".input_field").keyup(function () {

        let id = $(this).attr("id");
        let input_number = id.split("_")[1]

        $(".rep_input").eq(input_number).html($(this).val());
    })

})
```

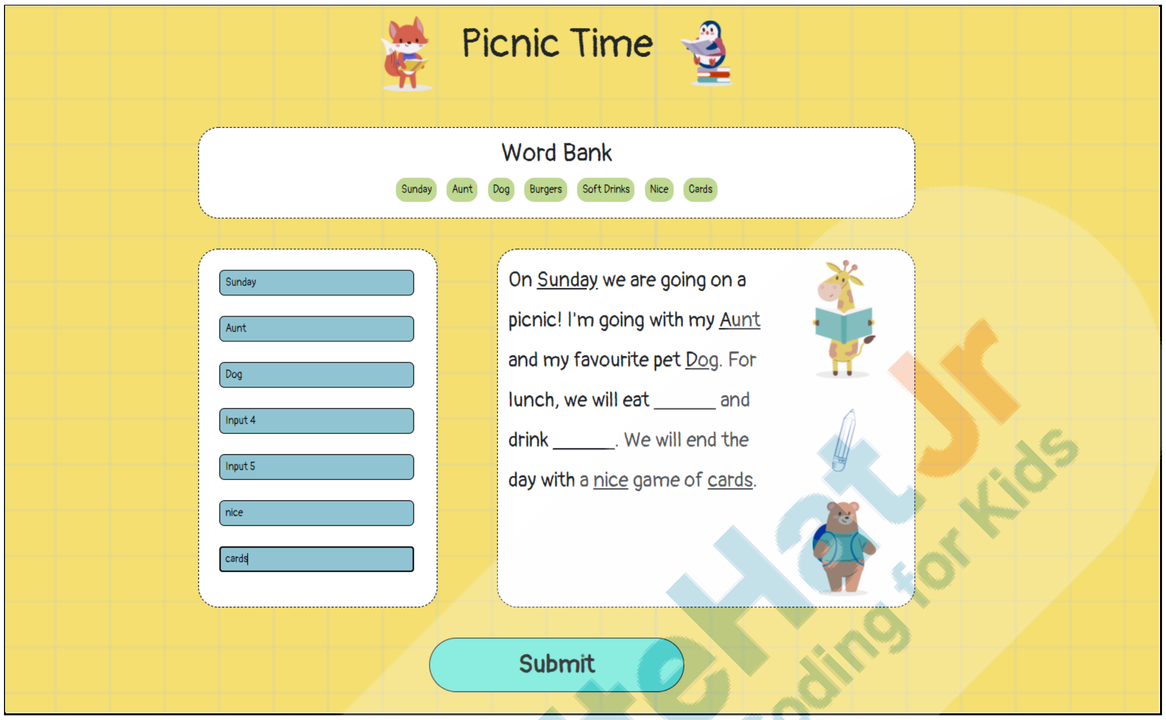
Similarly, we can add the **click event** on the button with id **#next_story** as well, and call the **displayStory()** function inside this listener function.



```
$(function () {
    $(".input_field").keyup(function () {

        let id = $(this).attr("id");
        let input_number = id.split("_")[1]

        $(".rep_input").eq(input_number).html($(this).val());
    })

    $("#next_story").click(function () {
        displayStory();
    })
})
```

With this, our functionality is complete. We can test it by typing into different input fields and seeing how it is updating the blanks of the story in real time.

*Guide the student to test the output.*



On Sunday we are going on a picnic! I'm going with my Aunt and my favourite pet Dog. For lunch, we will eat ______ and drink ______. We will end the day with a nice game of cards.

**Teacher Guides Student to Stop Screen Share**

**WRAP UP SESSION - 5 mins**

**Teacher Starts Slideshow**
**Slide 25 to 30**

**Activity details**
**Following are the WRAP-UP session deliverables:**
- Appreciate the student.
- Revise the current class activities.
- Discuss the quizzes.

**WRAP-UP QUIZ**
Click on In-Class Quiz

## Continue WRAP-UP Session
**Slide 31 to 36**

**Activity Details**

**Following are the session deliverables:**

- Explain the facts and trivia
- Next class challenge
- Project for the day
- Additional Activity (Optional)

**<u>FEEDBACK</u>**

- **Appreciate and compliment the student for trying to learn a difficult concept.**
- **Get to know how they are feeling after the session.**
- **Review and check their understanding.**

| Teacher Action | Student Action |
|---|---|
| You get Hats off for your excellent work! | *Make sure you have given at least 2 Hats Off during the class for:* |

**PROJECT OVERVIEW DISCUSSION**
Refer the document below in Activity Links Sections

| | | |
|---|---|---|
| | **Teacher Clicks** ✖ End Class | |
| **Additional Activities** | *Encourage the student to write reflection notes in their reflection journal using markdown.*<br><br>Use these as guiding questions:<br>● What happened today?<br>○ Describe what happened.<br>○ The code I wrote.<br>● How did I feel after the class?<br>● What have I learned about programming and developing games?<br>● What aspects of the class helped me? What did I find difficult? | *The student uses the markdown editor to write their reflections in a reflection journal.* |

| **Activity** | **Activity Name** | **Links** |
|---|---|---|
| Teacher Activity 1 | Boilerplate Code | https://github.com/whitehatjr/PRO-C176-Boilerplate-Teacher |
| Teacher Activity 2 | Teacher Reference Code | https://github.com/whitehatjr/PRO-C176-Code-Ref |
| Teacher Activity 3 | jQuery Syntax Reference Document | https://obj.whitehatjr.com/b29e1e1f-2882-4937-b7bb-64bf09c85d00.pdf |
| Student Activity 1 | Boilerplate Code | https://github.com/whitehatjr/PRO-C176-Boilerplate-Student |
| Teacher Reference 1 | Project Document | https://s3-whjr-curriculum-uploads.whjr.online/cf02f557-c527-44c6-8b63-f6f2bb05bc03.pdf |
| Teacher | Project Solution | https://github.com/whitehatjr/PRO-C176-Pro |

| | | |
|---|---|---|
| Reference 2 | | ject-Solution |
| Teacher Reference 3 | Visual-Aid | https://s3-whjr-curriculum-uploads.whjr.online/a7fffa3e-6932-4a87-b3fb-c5da8c41bac9.html |
| Teacher Reference 4 | In-Class Quiz | https://s3-whjr-curriculum-uploads.whjr.online/286c5447-bc80-455d-b803-968b6e1121f8.pdf |