JS 22 Calendar OOP

So far, we have done too many valid input checks. Is there an easier way? Yes, there is and the solution is to limit the number of values that the user can enter. By doing so, we only give the user valid input and don't need to filter the input.

What about the situation where the event has to be moved to a different day? We accounted for this in JS 21. What if the user wants to move an event to a **different day AND a different month**? We will use JS 21 as a template on how to move an event to a different day and month.

In this session, we continue to focus on **admin control** by only giving acceptable values that can be chosen from and the movement of an event to a **different day AND a different month**.

1. Add and remove 2. Options for users

==========

1 . **Write The HTML Code In Between** `<body> </body>.` Write the code in **BLUE and PURPLE** and position it **ABOVE** the code in **BLACK.**

```
<select id="dMenu">

      <option id = "d_1"  value="1" selected > 1   </option>
      <option id = "d_2"  value="2"> 2   </option>
      <option id = "d_3"  value="3"> 3   </option>


      <option id = "d_4"  value="4"> 4   </option>
      <option id = "d_5"  value="5"> 5   </option>


</select>
```

```
<select id="mMenu">
      <option id = "m_1" value="jan_events" selected> Jan   </option>
      <option id = "m_2" value="feb_events"> Feb   </option>
      <option id = "m_3" value="mar_events"> Mar   </option>

</select>
```

Continue to the next page

Explanation

In **JS 8**, we used radio boxes to allow selection of only a single choice. The private attribute "**checked**" can be used to determine which radio box was chosen by the user. If "**checked**" has the value of "**true**", then the user clicked on that specific radio button. Since **ONLY ONE** radio button can be "**checked**", this means only one conditional statement below can be true.

```
function get_cc( uRB, hRB )
{
	var cc = "#606060";
	if( uRB.checked == true )
	{
		cc = "#ff0000";

	}
	else if( hRB.checked == true )
	{
		cc = "#483D8B";
	}
	else
	{
		cc = "#a0a0a0";
	}

	return cc;

}
```

We now transition to the `<select>` tag that is used to give multiple options **AND** can serve as our input filter. The best way to filter something is to only give **VALID** input as the selectable options. This is why `<select>` is so important!!!

The **options** given to the user are **ALREADY VALID** and only a **SINGLE** valid option can be selected. This a two for one where we give freedom by giving multiple options and the options given are already valid and no input filter is needed.

How do we know which option was "**selected**" by the user? The "**selected**" is a private attribute and we can use the member access operator to access it. Next, we check it is true. The **GOLD,** JS code in **Chapter 3** will perform the check.

2. **Write The HTML Code In Between** <body> </body>. Write the code in **BLUE and PURPLE** and position it **ABOVE** the code in **ORANGE.**

```
<th
id      = "t0"
onclick = "tabSel( 0 )"
>
      Jan <span id = "jan_events_ecount"> </span><br/>
      <span id = "jan_events_urCount"> </span> <br/>
      <span id = "jan_events_hiCount"> </span>
</th>
```

Explanation

The code in **BLUE** is a special HTML tag that is used to indicate sections (very, very, very small sections). Notice that there is no data in between the open and closing span tags. Later on, we will update the in-between with the number of events and urgent events in January by using JS code.

The important part is the id, which is used to identify an HTML element by giving it a name. Remember that **JS code links up with HTML** by using
1. The id of the HTML element
2. `document.getElementById();`

The **
** tag means "**break**" and it is equivalent to pressing "enter" to go to the next line.

What is the difference between and ? The tag is for small sections while is for large sections.

> If you want to break up a website into **large sections**, then use ** .**
>
> However, if you only need **very, very, very, very small sections**, like a couple of words, then use .

3. **Write The JS Code In Between <script> </script>. Write all of it!!! Color code is for explanation.** HINT: get_pLevel()and checkpoint_pLevel() are private functions of EventProf … where do you put the code?

```
get_pLevel()
{

	// -- determine priority, id start at 0
	var pLevel = 0,
	found      = 0,
	i          = 0;

	do
	{
		var pObj = document.getElementById( "pri_" + i );

		if( pObj.selected == true )
		 {
			found = 1;
		}
		 else
		 {
			i++;
		}
	}
	while( i < 3 && !found );

	// -- store new position into private attribute
	this.pLevel = i;

}


checkpoint_pLevel()
{
	var
	i          = 0;     // -- pLevel starts at 0

	// -- loop through and reset
	do
	{
		var pObj = document.getElementById( "pri_" + i );

		pObj.selected == false;

	}
	while( i <= 3 );

	// -- checkpoint
	document.getElementById( "pri_" + this.pLevel ).selected = true;

}
```

Continue to the next page

Explanation

Let's start with the function definition of `checkpoint_pLevel()`. This function definition is called when the user wants to modify an **EXISTING** event. We are using a unique loop here called the do-while loop. **Notice that the semi-colon is after the while() statement.**

In all other loops, we check the loop condition first and run the loop body second. **However, the do{ } while(); is the opposite** – we run the loop body first and check the loop condition second. A do-while(); loop is for single pass first and check second.

A good example is the function definition `get_pLevel()`, which is above. What if we find that the user selected the first option for priority level, which is "urgent"? If we found it already, do we still need to run the loop? No we don't. So, we can stop since the loop condition is already satisfied.

However, after the single pass and we find that the user selected the third option, then the loop continues. So, the do-while() loop saves us time since it exits if the user chose the first option and continues if the user chose any other option.

4. **Write The JS Code In Between `<script> </script>`. ONLY Write all of it!!! Color code is for explanation.** HINT: `update()` is a **private function** of EventProf … where do you put the code?

```
update()
{
	alert( "inside private fx update()" );

	var msg		= document.getElementById("nEInput").value;
	var title	= document.getElementById("nETitle").value;

	alert( "please work" );

	// --
	this.get_day();
	var day		= this.day;

	this.get_month();
	var month	= this.tabId;

	this.get_pLevel();
	this.get_cc();

	alert( "day: " + day );

	 // -- same day, just update data
	if( day == this.prevDay )
	{
		alert( "same day" );
```

```
        // -- same day AND same month
        if( month == this.prevMonth )
        {
                this.eColor = this.get_cc(  );
                this.msg    = msg;
                this.day    = day;
                this.eTitle = title;

      var eObj = document.getElementById( this.tabId + "_" + this.day + "_" + this.eTicket );

                alert( "this.eColor: " + this.eColor );
                console.log( "this.eColor: " + this.eColor );
                eObj.style.backgroundColor = this.eColor;

                eObj.innerHTML = this.eTitle;

        }
         // -- same day BUT different month
        else
        {
                // -- remove old row
  var oldObj = document.getElementById( this.prevMonth + "_" + this.day + "_" + this.eTicket );

                oldObj.remove();

                 //this.go_to_IT();
                 //month_count_calc( this.prevMonth );


                this.tabId  = month;
                this.eColor = this.get_cc( );
                this.msg    = msg;
                this.day    = day;
                this.eTitle = title;

                // -- make a new row
                this.make_clone( this.eTicket );

                // --
                this.hide_pop_up();

                /* -- new location means new HTML element was added on the fly
                a. add event listener on this new location
                */
                this.set_event_listener( this.eTitle );

                //this.go_to_IT();
                //month_count_calc( this.tabId );

        }

}
// -- different day, remove old row and put data in new row
else
{
        alert( "different day" );

        // -- different day BUT same month
```

```
        if( month == this.prevMonth )
        {
                // -- remove old row
var oldObj = document.getElementById( this.tabId + "_" + this.prevDay + "_" + this.eTicket );

                oldObj.remove();

                this.eColor = this.get_cc( );
                this.msg    = msg;
                this.day    = day;
                this.eTitle = title;

                // -- make a new row
                this.make_clone( this.eTicket );

                // --
                this.hide_pop_up();

                /* -- new location means new HTML element was added on the fly
                a. add event listener on this new location
                */
                this.set_event_listener( this.eTitle );
        }

        // -- different day AND different month
        else
        {
                // -- remove old row
var oldObj = document.getElementById( this.prevMonth + "_" + this.prevDay + "_" + this.eTicket );

                oldObj.remove();

                //this.go_to_IT();
                //month_count_calc( this.prevMonth );


                this.tabId  = month;
                this.eColor = this.get_cc( );
                this.msg    = msg;
                this.day    = day;
                this.eTitle = title;

                // -- make a new row
                this.make_clone( this.eTicket );

                // --
                this.hide_pop_up();

                /* -- new location means new HTML element was added on the fly
                a. add event listener on this new location
                */
                this.set_event_listener( this.eTitle );

                //this.go_to_IT();
                //month_count_calc( this.tabId );

        }
```

```
	}
}
```

Explanation

There are several situations that we must account for.
1. The event title and message were updated to something else
2. The event is **moved** to a different day
3. The event is **moved** to a different month
4. The event is **moved** to a different DAY **and** MONTH

In situation 2 – 4, we are moving an event to a different row (where each row is a different day) or a different month (where each tab is a different month). This means that we have to delete the existing event block and make a new one.

So, look at the code in **ORANGE** and see that we have to remove the old event object. Next, we make a new clone and put the new clone in a different day or month.
1. The steps above creates the effect of "moving an event block" by simply repositioning it in a different location

5. **Write The JS Code In Between <script> </script>. WRITE ALL OF IT!!! Color code is for explanation.** HINT: function month_count_calc() is a public function … where do you put the code?

```
function month_count_calc( monthId )
{
	console.log( "inside month_count_calc()" );

	// -- using repository
	var mObj   = document.getElementById( monthId + "_ecount" );
	var mUrObj = document.getElementById( monthId + "_urCount" );

	var
	totalCount  = 0,
	urCount     = 0;

	for( var i = 0; i < eArr.length; i++ )
	{

		// -- check if month matches parameter
		if( eArr[i].tabId == monthId )
		{
			totalCount++;
			if( eArr[i].priLevel == 0  )
			{
```

```
                        urCount++;

                    }
                }

        }

        mObj.innerHTML   = "(" + totalCount + ")";
        mUrObj.innerHTML = "( U: " + urCount + " ) ";

}
```

Explanation

The code above is a function definition. Remember that a function definition is giving code a name so that we can reuse it later on. The name of the code above is **`month_count_calc().`**

The code in **PURPLE** matches with the id of chapter 2. This means that the code above is using a "**for**" loop to access the private attribute of `tabId` to compare it with the input parameter `monthId`.

So, if we want to find out the number of events in the month of January, then the data "**jan_events**" is stored into the input parameter `monthId`. Next, we execute the first "**if**".

An event in January means that the first "**if**" statement is true and we increment the variable `totalCount` by 1.

However, if the event object **IS NOT** in January, then the "**if**" statement is false.
1. Since there is no "**ELSE**" statement, a false means we do nothing.

> The **code above** is the **function definition**. When do we **use** the function definition? **We have to call the name of the function definition in order to use it.**
>
> Notice that the **code below** in Chapter 6 is the function call (**ie. we are calling the name of the function definition**) and we pass in the month as the input parameter.

6. **Write The JS Code In Between <script> </script>. WRITE ALL OF IT!!!**
Color code is for explanation.

```
// --
function mod_eObj()
{
	alert( "inside  mod_eObj() " );

	//this.tabId + "_" + this.day + "_" + this.eTicket;

	// var objToMod = document.getElementById( chosenEId );
	eArr[chosenTicket].update();

	month_count_calc( "jan_events" );
	month_count_calc ( "feb_events" );
	month_count_calc ( "mar_events" );

}
```

Explanation

The code in **GOLD** is the function name that was written in Chapter 5. Anytime we write the function name, we are call it and using it.

The **first GOLD** code wants to know how many events are in January and so we give "`jan_events`" as an input parameter. If we **look back at section 5**, the input "`jan_events`" is stored into `monthId`

Test

Click on the **GREEN** "**Run**" button and do the following

1. Click on the button "**Add New Event**" and the website will create a **RED** event object on the Jan tab of day 1.
2. Click on the **RED** event block of Jan 1 to activate the pop up banner. Remember, the pop up banner is in "**toggle**" mode → clicking on an event block will make it appear or disappear.
3. When the pop up banner appears, go up to the select menu and change the month to Feb and the day to 5
4. Next, click on the **white** "**Mod**" button
5. **Effect:** The event block should move from Jan 1 to Feb 5

JS Challenge

Currently, we have a count of the **number of urgent events**. What about the number of **high priority events?**

Look at the **code of section 5** and see that we have two variables named `totalCount` and `urCount`.

1. Make another variable called `hiCount` and load the data 0 into it.

Next, update the code in Chapter 5 to **determine the number of high priority events**.

HINT:

a. You will need another "`if`" statement to perform the check
b. Use square bracket notation of the array ([ ]) and the member access operator to access the private attribute of `priLevel`
c. If **urgent priority** has a `priLevel` of 0, then **high priority** has a `priLevel` of …