Controlling Details on the Pages

Using Custom Buttons

Right now our consent app requires subjects to click "I consent" and then click the submit button. It gets the job done but it should be enough to consent by just clicking once. Go to the consent app and open ConsentPage.html. Delete the formfields and nextbutton commands. Next, insert a button by using the standard html

```
<button> </button>
```

command. It's very easy to submit data using a button. Simply name the button like the model that we are using ('consent') and specify the value (1, i.e. True). Then, add some text in the command to give the button some text.

```
<button name='consent' value=1>I consent</button>
```

But watch out! By default, all buttons on the page submit the form. If you want to include a button that should not submit the page, specify that by adding the type 'button'.

```
<button type='button'>I am a red herring</button>
```

Test it in otree running devserver. It works -- but it is really ugly. We can do better than that. Luckily, oTree already uses the bootstrap library which is a standard library for things like buttons in web development. You can look at the options for Buttons (and other menus) here. https://getbootstrap.com/docs/5.0/components/buttons/

To use a bootstrap button we simply call the bootstrap class 'btn btn-primary'.

```
<button name='consent' value=1 class='btn btn-primary'>I consent</button>
<button type='button' class='btn btn-primary'>I am a red herring</button>
```

Using div and styles

In a simple experiment it is fine if we simply have some text and some input all in one long column. But for some experiment we want more control. In html we can use the

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

command to separate the page into different elements and we can control the location of those divs using a grid that we set up using the

```
<style> </style>
```

command. Let's set something like this up in the public goods game app in Feedback.html. We will use this page later to set up some graphical feedback and we can use the divs to set up a structure. Let's create a setup that looks as follows

```
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
YYYYYYYYY ZZZZZZZZ
YYYYYYYYY ZZZZZZZZ
YYYYYYYYY ZZZZZZZZ
```

Where X is a div for a graph, and Y and Z are divs for some text.

oTree supports bootstrap and bootstrap has a lot of handy systems for this.
https://getbootstrap.com/docs/5.0/layout/grid/

We start by setting up a high-level container div that contains all the elements and centers the text inside.

```
<div class='container text-center'>
</div>
```

Within this container we now place a first div of the row class

```
<div class='container text-center'>
<div class='row'>
</div>
</div>
```

Within this row we set up a div of a col class.

```
<div class='container text-center'>
<div class='row'>
<div class='col'>
Top
</div>
</div>
</div>
```

Next, we start a new row and fill it with two more columns.

```
<div class='container text-center'>

<div class='row'>
<div class='col'>
Top
</div>
</div>

<div class='row'>
<div class='col'>
Bottom Left
</div>
<div class='col'>
Bottom Right
</div>
</div>

</div>
```

Note how this setup mimics the table we set up earlier row by row and cell by cell. We can replace the paragraphs we had on this page. Let's put a small note on total contributions in the bottom left div and detailed contributions in the bottom right div.

```
<div class='container text-center'>

    <div class='row'>
    <div class='col'>
    Top
    </div>
    </div>

    <div class='row'>
    <div class='col'>
        <p>In total, all players in the group contributed {{ group.total }}
to the group account.</p>
    </div>
    <div class='col'>
        <p>You contributed {{ player.contribution }} and you earned {{
player.payoff }}.</p>
        <p>Partner 1 contributed {{ others_contributions.0 }} and earned {{
others_payoffs.0 }}.</p>
        <p>Partner 2 contributed {{ others_contributions.1 }} and earned {{
others_payoffs.1 }}.</p>
        </div>
    </div>

</div>
```

Not too pretty if you check it out, but you get the idea.

To control how the rows and cells behave we can set up a new block for styles.

```
{{ block styles }}

{{ endblock }}
```

In this block we can set up

```
<style></style>
```

commands to control elements using css commands. For example, we can add margins on top and bottom of all rows.

```
{{ block styles }}
<style>
.row {
    margin-top: 10px;
    margin-bottom: 10px;
}
.col {
    margin-left: 10px;
    margin-right: 10px;
}
</style>
{{ endblock }}
```

There are lots of options and existing examples for bootstrap pages, so if you want to set up a page looking in a specific way you can often find html code snippets for bootstrap that get you started. All in all I won't do a full html and css class but these options should give you an idea of how the systems work and how to get started.

Simple Javascript Button

Javascript allows us to program interactive html pages, for example hiding/revealing elements with buttons, custom input widgets or graphical feedback/intercaces. The drawback is that is javascript is yet another programming language and, unlike Python, javascript is very cumbersome. The advantage is that it is widely used in website development and there are many example of working code online. If you want to implement something on your page instead of starting from scratch you can start by googling. To implement a button, for example you could search for "javascript button reveal stackoverflow" for a discussion thread on stackoverflow or "javascript button reveal jsfiddle" for a working example on jsfiddle.

Let's create our own button in the pgg app. We will hide the budget information by default but give subjects a button to reveal the information how much budget they have. Head to pgg to Choice.html and find the paragraph that we included for the budget information. Add two pieces of information here: set the style to hidden and add an id. We will need the id to manipulate its attributes.

```
<p style='visibility: hidden' id='budget_paragraph'>
    You have a budget of {{ C.BUDGET }}.
</p>
```

Next, add a button that simply says show my budget and use a standard bootstrap button. Add an id for the button and don't forget to use set the type to button so the button does not submit the page.

```
<button id='info_button' type='button' class='btn btn-primary'>
    Show my Budget
</button>
```

Now, we need to add some javascript. Add a new block for scripts. Within this block, start and add a script.

```
{{ block scripts }}
<script>

</script>
{{ endblock }}
```

Within this script, first we define a function that reveals the paragraph by changing its visibility attribute from hidden to visible but only if the the paragraph is visible. So when the function is called, first we check if the element is hidden. Access the html document by "document", get the paragraph by finding the element by id and check the style attribute. I use triple equal signs here for an exact check. You could also use double equal signs for a non-exact check (which allows zero to be equal to true, for example).

```
function reveal() {
    if (document.getElementById('budget_paragraph').style.visibility ===
'hidden') {
        document.getElementById('budget_paragraph').style.visibility =
'visible';
    } else {}
}
```

Afterwards, we simply have to map the function to the button. We can simply add a listener that executes a function as soon as the event "click" (as in the participant clicks on the button) happens.

```
document.getElementById('info_button').addEventListener('click', reveal);
```

Putting it all together

```
{{ block scripts }}
<script>
function reveal() {
    if (document.getElementById('budget_paragraph').style.visibility ===
'hidden') {
        document.getElementById('budget_paragraph').style.visibility =
'visible';
    } else {}
}
document.getElementById('info_button').addEventListener('click', reveal);
</script>
{{ endblock }}
```

Custom Slider

Let's add another javascript element and build a custom slider for the contribution decision in the public goods game. Subjects will be able to use the slider to make their decision how much to contribute to the group account. Remember that we have a budget that is defined in the constants -- we will have to tell the slider how much the budget actually is. In html we were able to access this information using the {{ C.BUDGET }} command but to access information in javascript we have to actively push the information to javascript. Go to pgg's init.py and add a function to the Choice page that pushes a dictionary to javascript. There is a pre-defined js_vars function for this purpose.

```
class Choice(Page):
    form_model = 'player'
    form_fields = ['contribution']

    @staticmethod
    def js_vars(player: Player):
        return dict(
            budget = C.BUDGET,
        )
```

Next go to Choice.html and delete the formfield command. First, add a paragraph that spells out how much the player is contributing to the group account. If you want to manipulate some text it is a good idea to use a span command that lets you manipulate the text within. Give the span an id so we can manipulate it.

```
<p> You are putting <span id='sliderValue'></span> into the group account.
</p>
```

Below, add a new div that will be the container for our slider. Again, we can use a bootstrap default for this. https://getbootstrap.com/docs/5.0/forms/range/

Use the command input and set the input up as a range that uses the bootstrap class form-range. You can also specify a minimum, maximum and the step size. Don't forget an id! As far as I know we cannot directly use the budget variable in this input here directly, so we will set it later using javascript.

```
<input type='range' class='form-range' min='0' value='10' step='0.5'
id='myContribution'>
```

We now have a slider that subjects can interact with but we don't store the value and we do not return the choice to oTree. To do this we have to set up a hidden input that is named just as the variable we are trying to store.

```
<input type='hidden' name='contribution' id='id_contribution'>
```

In the scripts block add a new script. First, gather all the elements that we are using in variables and define a function that updates both the text and the data that is passed to oTree. As soon as the page is loaded and the script is executed, update once and also update the slider's maximum with the javascript variable. Afterwards, add a listener to the slider that executes the update function as soon as the participant interacts with the slider.

```
<script>
var slider = document.getElementById('myContribution');
var output = document.getElementById('sliderValue');
var data = document.getElementById('id_contribution');

function update() {
    output.innerHTML = slider.value;
    data.value = slider.value;
}

slider.max = js_vars.budget;
update();

slider.addEventListener('input', update);
</script>
```