Our First App

oTree Structure

In oTree the basic building block of an experiment is an "app". Basically, it is a self-contained game or task. It is possible to pass data between apps, but it is a bit more tedious. A typical setup of an oTree experiment has the structure

Consent -> Game -> Questionnaire -> Payment Information

oTree follows the Python logic of structuring objects hierarchically. If we run an experimental session, oTree creates a "Session". Each Session has a number of "Participants" attached, which correspond to the subjects in our experiment. Each Session is split into a series of "Subsessions". These Subsessions are instances of the apps that we set up. If we repeat the game two times, we end up with the following series of Subsessions:

Consent -> Game[Round 1] -> Game[Round 2] -> Questionnaire -> Payment Information

Within each Subsession, Participants are assigned to "Players" and "Groups". Within each Subsession, Player and Group assignment is stable, but between Subsessions they change. So in Game[Round 1], Participant 1 may be player 3 who is part of group 1, but in Game[Round 2] he may be player 1 who is part of group 2.

Consent App

Let's start with a standard reusable app, a consent page. Before participating in our experiment, subjects have to give their consent. Open the terminal, make sure that you are in the oTree folder and type

```
otree startapp consent
```

In the Explorer tab on the left, find and open the consent folder. The new app has three files: __init__.py and two .html files. The former defines the experimental procedures in Python language, while the latter defines the websites that participants get to see.

Setting up __init.py__

Open __init.py__. The first few lines are import functions that import code from the oTree package. Later we will add other packages, but we don't need any fancy packages for the consent app. Next is the documentation. Here you can add a short description of what the app does.

The next paragraph sets up the constants for the app. The first line, NAME_IN_URL governs what the url in the browser will be for this app. This has to be unique, so no two apps may share the same app name in a session. Also, consider that in an online experiments subjects may see the name in their browser's address bar, so refrain from using loaded language. 'consent' is fine for our purposes. PLAYERS_PER_GROUP defines how many players are in a group. A consent app has no groups so we leave it at None. Finally, NUM_ROUNDS sets up how often an app is repeated. 1 is fine for a consent app. Later on we will set up our own constants in the same manner.

Below you can define variables for subsessions, groups and players. Note that pass means skip in Python. In our app we want to record the *player's* consent. oTree comes with a range of different fields, which they call models.

- BooleanField() for true/false values
- FloatField() for real numbers
- CurrencyField() for currency amounts
- IntegerField() for integers
- StringFields() for text
- LongStringField() for long text (it basically gives you a bigger field to write in)

https://otree.readthedocs.io/en/latest/models.html#fields

So we go to the Player class and add a new variable called consent. Mind the indent after Player! Python structures code by indentation. All the code that belongs to the Player class is indented by one tab (or four blank spaces). If you set up your indents incorrectly, the code will not work. Also, leave two empty lines between each class so it is clear where one class definition starts and the next begins. Finally, let's leave a comment here. Anything after # in a line is recognized as a comment by Python. It is always a good idea to comment your code so you will understand what you did when you programmed the experiment.

```
class Player(BasePlayer):
    consent = models.BooleanField() # this variable records if the subject
gave informed consent
```

Continue downwards. oTree automatically created classes for MyPage, ResultsWaitPage and Results. We want our own page which we will call ConsentPage. We could delete the automatically created classes but it does not hurt to leave them in. After the Results class (keep two empty lines!) add the new class ConsentPage. We tell oTree that this is a standard page, so it extends the pre-defined oTree Page class. On this page we expect some input. The player will give consent, so we tell oTree to expect the form_model player and to expect the field input for consent. Note that the latter has brackets -- this is the Python way of defining a list. In our case there is only one variable, but we could have multiple variables on the same page.

```
class ConsentPage(Page):
    form_model = 'player'
    form_fields = ['consent']
```

Scroll down to page_sequence. This variable defines the order of the pages that are displayed to subjects. Our consent app is not an interactive game so there is no need to wait for the other players. The app has a single page, the ConsentPage.

```
page_sequence = [ConsentPage]
```

Setting up ConsentPage.html

We told the oTree logic what kind of data to expect on the consent page. Now we need to set up the actual page that subjects will interact with. Create a new file in the consent folder by right-clicking the consent folder and name the file ConsentPage.html. An oTree page is divided into a few basic blocks. The title block sets the title of the page. The content block sets up the main page. Finally, later on we will look at the scripts block that allows for interactive graphics.

Let's start with the title block. Let's give the page a simple title. Note that html does not require you to indent your code like Python does. Still it can be useful to structure

```
{{ block title }}
    Consent Form
{{ endblock }}
```

Next we add a content block below the consent block. The content block should contain some text, the form and a button to submit the form. Within a block we can use standard html code. In html we can create a paragraph by using the command. Below the text the form should appear. The {{ formfields }} command automatically creates forms for all the formfields that we defined in the ConsentPage class. Later on we will look at specific fields in specific places. Finally, we use the {{ next_button }} command to tell oTree where to put the button to submit the current page.

```
{{ block content }}
    <p>To participate in this experiment you have to consent.</p>
    {{ formfields }}
    {{ next_button }}
{{ endblock }}
```

Defining a Session Config

We have our first app all set up. Now we need to set up a session config that uses this app. In the main oTree folder, navigate to settings.py. This is the main configuration file for your experiments. Close to the top you can find the line SESSION_CONFIGS. This config is a list of all the configurations for sessions that are defined for this oTree instance. Each config is a dictionary, and the configs are separated by commas. Because we included the sample games, there are already two session configs in there, one for a simple survey, and one for a guess-two-thirds game.

Each session config is a dictionary. In Python, dictionaries match some entry (on the left) with a value (on the right). Entries are separated by commas. We will add a new config below the existing configs. The name of the config has to be unique.

```
name='course',
```

display_name is the name that is displayed in the admin panel.

```
display_name='WZB oTree Course',
```

In the app_sequence we define which apps this session config is going to use. This is defined as a list of apps, referenced by their app name strings.

```
app_sequence=['consent'],
```

Finally, we have to define how many participants are automatically created if we run the app in demo mode. Demo mode is a good way to create a test session with one simple click in the admin panel. Let's use three demo participants.

```
num_demo_participants=3,
```

Putting it all together, the SESSION_CONFIGS should look as follows:

```
SESSION_CONFIGS = [
    dict(
        name='guess_two_thirds',
        display_name="Guess 2/3 of the Average",
        app_sequence=['guess_two_thirds', 'payment_info'],
        num_demo_participants=3,
    ),
    dict(
        name='survey', app_sequence=['survey', 'payment_info'],
num_demo_participants=1
    ),
    dict(
        name='course',
        disply_name='WZB oTree Course',
        app_sequence=['consent']
        num_demo_participants=3,
    ),
]
```

A First Test

Make sure that you are in the oTree folder and that the virtual environment is up. Open your Terminal and run a local server.

```
otree devserver
```

Navigate to http://localhost:8000 in your browser. You will start on the Demo page. If you click on 'WZB oTree Course' you will automatically start a demo session with three participants. Let's look around on this page. Initially, you start in the 'Links' tab where all the links that are used to join a session are available. In the 'Monitor' tab you can observe on which page participants currently are and if they are waiting for another participant. In the 'Data' tab you can take a live peek at the data as it is being generated. Finally, in the 'Payments' tab you will see participants' earnings.

Return to the 'Links' tab. There are two ways to join a session. If you click on the session-wide link, oTree will automatically fill in Participants P1, P2, P3 until all slots are filled. If you click this link a fourth time, you will get the feedback that the session is full. Alternatively, you can use the single-use links. Single-use links are tied to participants and they are robust to a participant closing the browser and reopening the link. oTree will recognize the participant and serve the correct current page. Also, there is no risk of multiple inputs if the same link is open in multiple pages, because oTree will disregard any input sent from a client from a page that oTree already recevied input for.

So let's have a look at the consent form. Title and paragraph look good, but the consent form is very basic. The label just says 'consent', and subjects have to option of not giving consent and still continuing with the experiment. Let's fix that.

Changing Labels and Choices

The oTree devserver command is very robust. We can change some simple things without the server crashing or a new session having to be created. It's very useful to build a page and immediately checking the output. So let the server running, return to __init__.py and locate the consent model in the Player class. We will feed some inputs, separated by commas, into the model: label and choices. label controls the text that subjects see in the form.

```
label='Do you consent?',
```

In choices we can define a list of choices that are available to subjects. BooleanField allows for True/False statements, but in the database it is saved as 1 (True) or 0 (False). If we feed any option other than 1/True or 0/False into BooleanField, like a string for example, the model will by default save 1 (True).

```
choices=[True, False],
```

We want to give subjects one option: to agree. So we could simply use

```
choices=[True],
```

but then the option would also show up as "True" in the form. To control the label of the option, we can use a list (within the list). The first element of the list controls the data that is saved in the database, while the second element is the displayed name.

```
choices=[[True, 'Yes']],
```

In the end, the Player class should look as follows.

```
class Player(BasePlayer):
    consent = models.BooleanField(
        label='Do you consent?',
        choices=[[True, 'Yes']],
    ) # this variable records if the subject gave informed consent
```

Open a new demo session (or reload an already open consent page) and you should see the changes immediately. When you are done, return to the terminal and stop the server by pressing Ctrl-C.