Philipp Chapkovski
Higher School of Economics, Moscow

oTree workshop

University of Zurich

Nov 6, 2018

About, Demo and plans for today

- Post-doc in International Laboratory for Experimental and Behavioural Economics (https://epee.hse.ru/en/, chair of Heike Hennig-Schmidt)

- https://zurich-workshop-2018.herokuapp.com/demo/
- !! Get familiar with oTree Admin

- Public good game with two treatments and interaction:
  - Heterogeneous endowments
  - Gender information about co-members

oTree: alternatives

- Lioness http://lioness-lab.org/ - G34, gm15, (Gaechter et al. 'Conducting interactive experiments online' 2018 *Experimental Economics*)
- NodeGame nodegame.org (G8, gm27, 47 Stars, 6 Forks on GitHub)
- Gorilla gorilla.sc, proprietary, see paper in *PsyArXiv* 2018
- Sophie https://www.sophielabs.com/ (gm48, Last release 2016)
- BreadBoard http://breadboard.yale.edu - Proto stage network project with Christakis (22 Stars, 7 Forks on GitHub)
- …
- oTree (www.otree.org), (G102, gm890, 219 Stars, 103 Forks on GitHub)

oTree

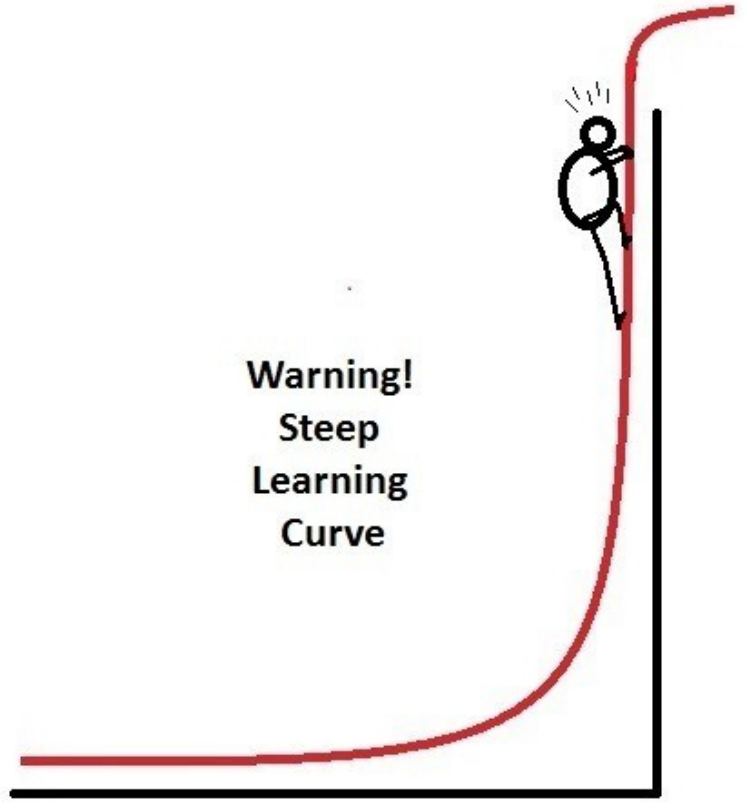
- platform to run interactive online experiments
- written on Python
- open source
- based on Django framework
- responsive – can be run in mobile browsers, tablets etc.

Code for this workshop:	https://github.com/chapkovski/zurich-workshop
My e-mail:	chapkovski@gmail.com
oTree support group:	https://groups.google.com/forum/#!forum/otree
oTree documentation:	http://otree.readthedocs.org/

oTree

- Languages: Python (backend); HTML, JavaScript, Django template language (frontend)
- Backend: **Django (M-V-C)**
- Frontend: **Bootstrap**

(https://getbootstrap.com)



Some more complicated examples

- Double auction
  - https://double-auction.herokuapp.com
- Employer-employee negotiations
  - https://negotiations-otree.herokuapp.com
- English auction
  - https://mini-ebay-otree.herokuapp.com/
- Information cascade
  - https://information-cascade-otree.herokuapp.com/
- Real effort tasks
  - https://morning-citadel-36858.herokuapp.com

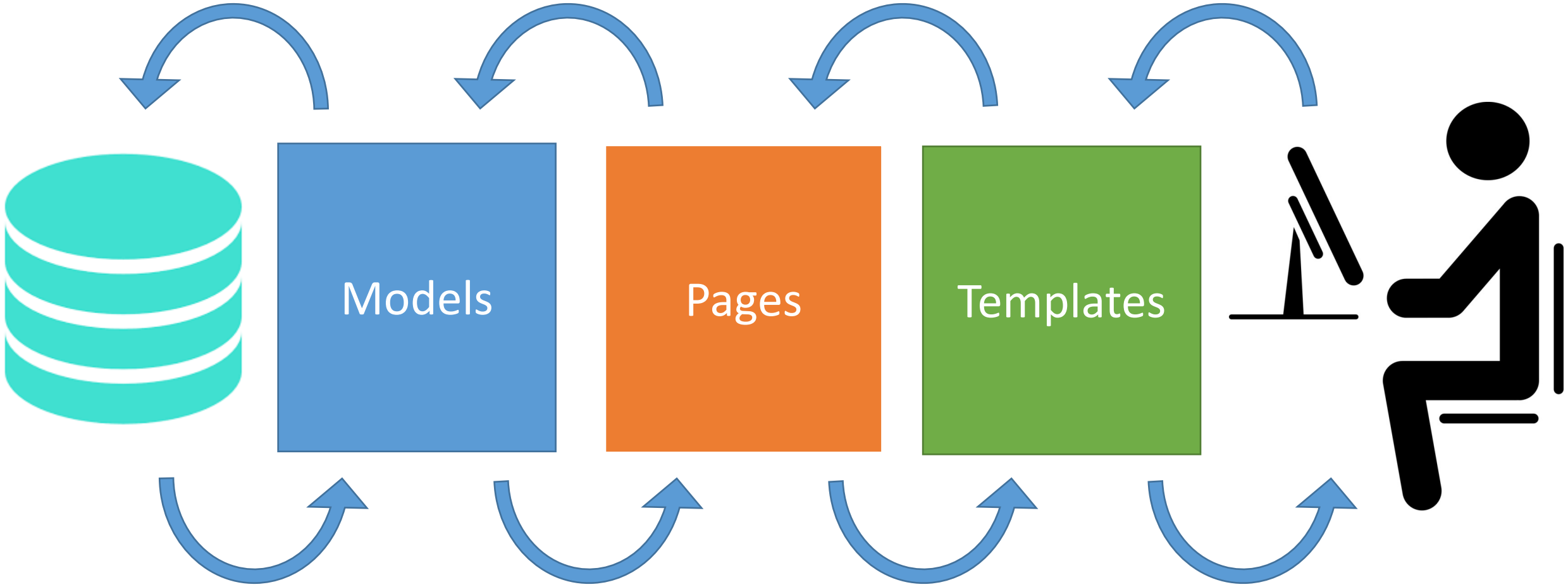
Let's create our first oTree app

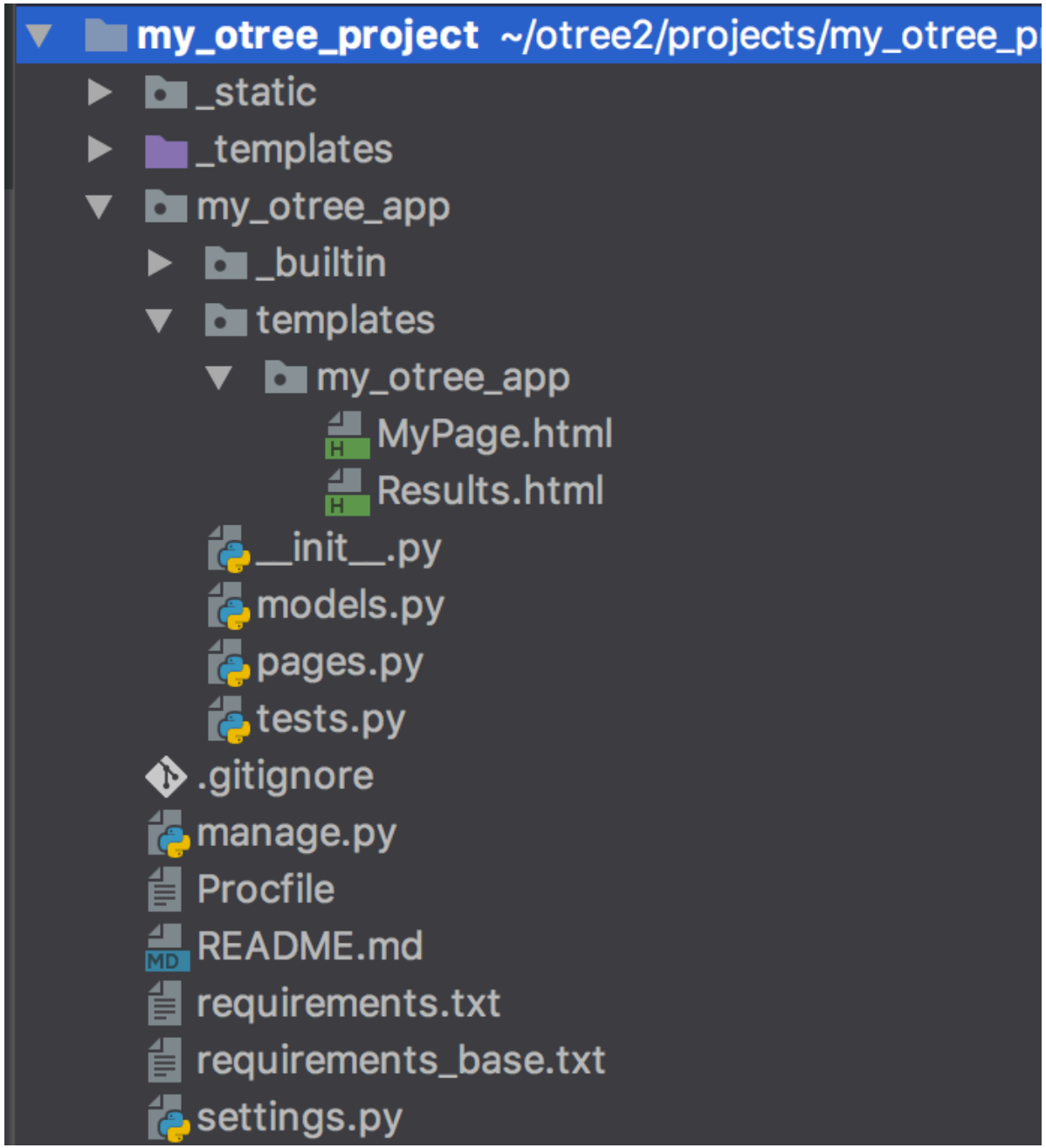
1. *RECOMMENDED: Create virtualenv*
2. **T:** `pip install -U otree`
3. **T:** `otree --version`
4. **T:** `otree startproject my_first_project`
5. **T:** `cd my_first_project`
6. **T:** `otree startapp my_first_app`
7. add app to settings
8. **T:** `otree devserver`
9. Browser: go to: http://127.0.0.1:8000

```
SESSION_CONFIGS = [
    {
        'name': 'my_first_app',
        'display_name': "My first oTree app",
        'num_demo_participants': 1,
        'app_sequence': ['my_first_app'],
    },

]
```

Our main goal:

1. Show something to the participant.
2. Get his/her/other reaction on it


Models
Pages
Templates

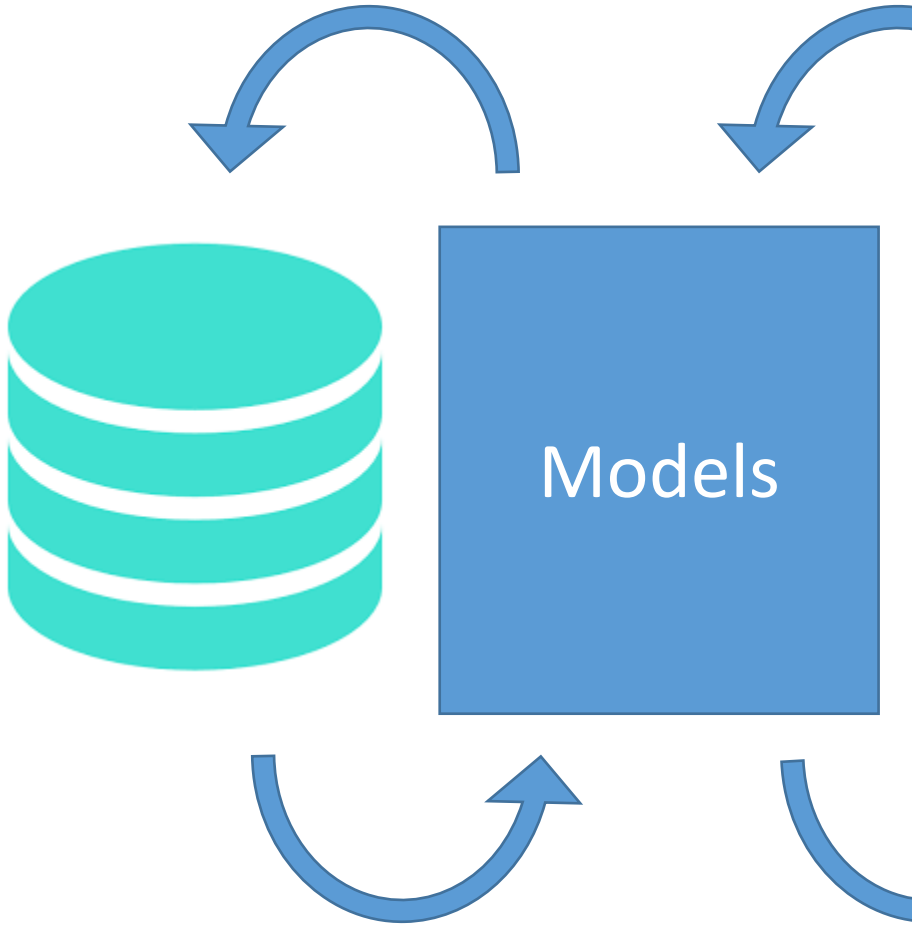

my_otree_project ~/otree2/projects/my_otree_p
_static
_templates
my_otree_app
_builtin
templates
my_otree_app
MyPage.html
Results.html
__init__.py
models.py
pages.py
tests.py
.gitignore
manage.py
Procfile
README.md
requirements.txt
requirements_base.txt
settings.py

Structure of oTree app

- Pages
  - `Page` classes
  - `page_sequence`
- Models
  - `Constants`
  - `Player, Group, Subsession`
- App definition in `settings.py`

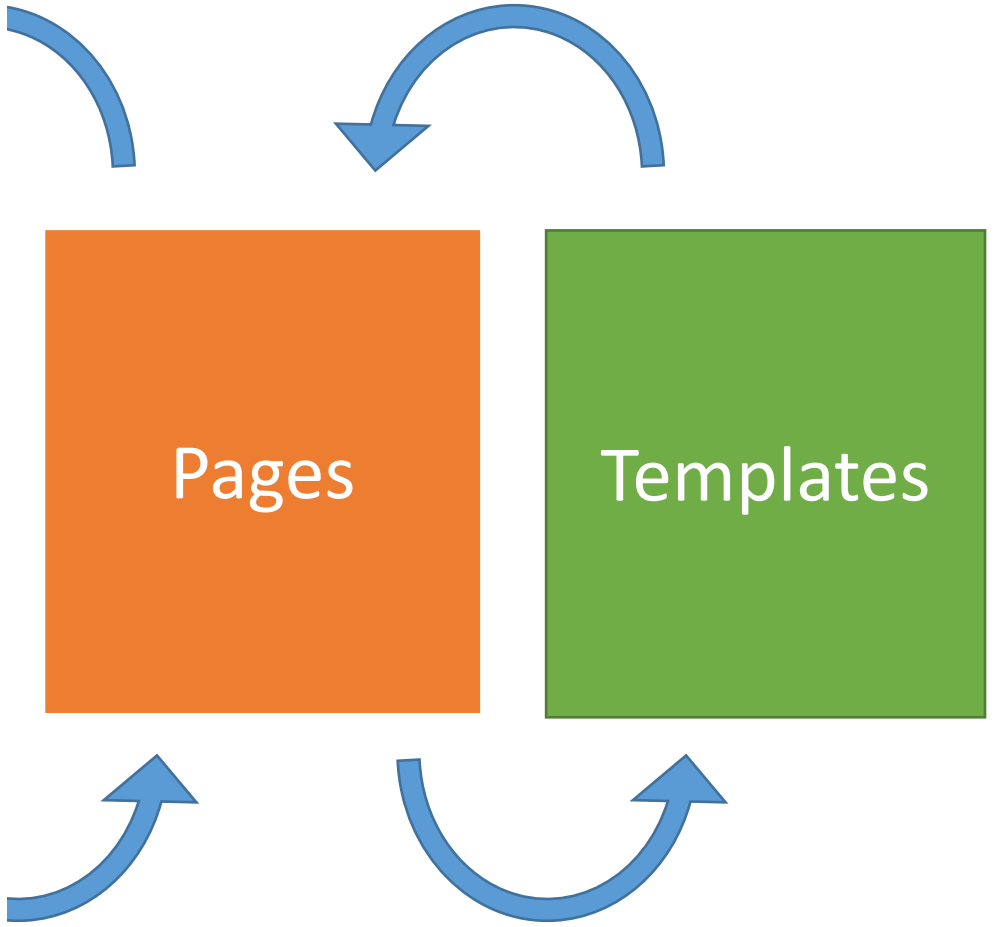
Displaying logic

Data processing

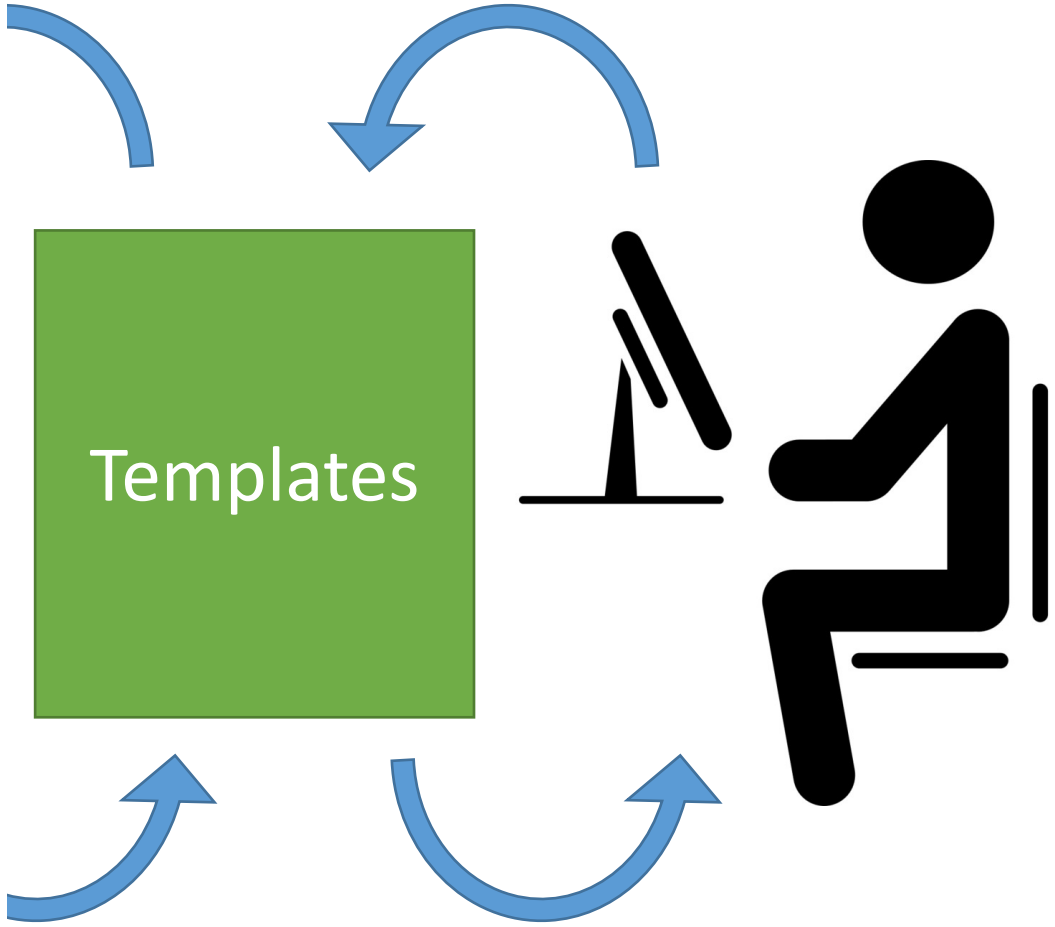


- Models are responsible for storing and processing data in the database
- Anything that you need to be stored should be defined as a field in **`models.py`** file

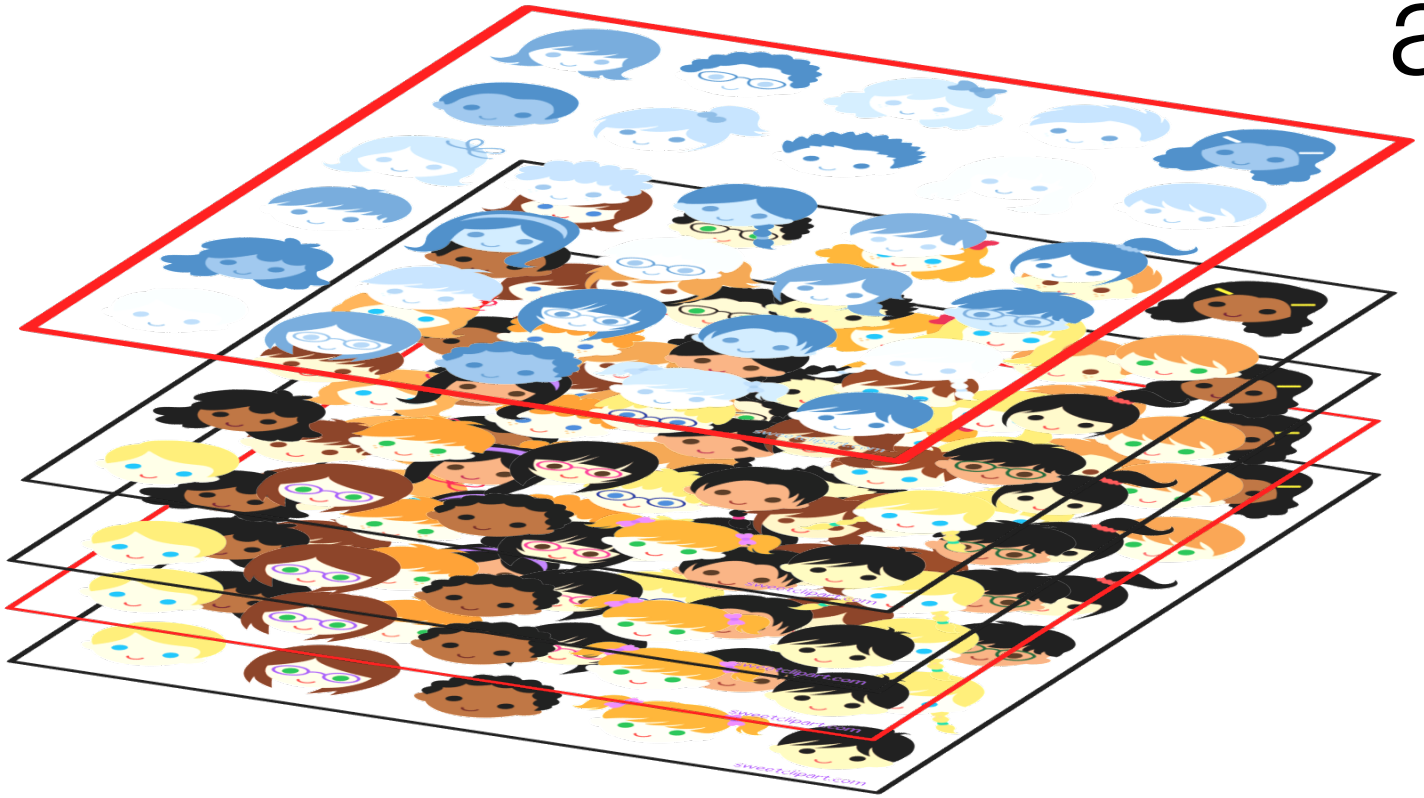
- Pages are responsible for retrieving and passing back data from **`models`** to templates and vice versa.
- If you need to show something to a participant or to get his/her input, you need to indicate this in **`pages.py`**



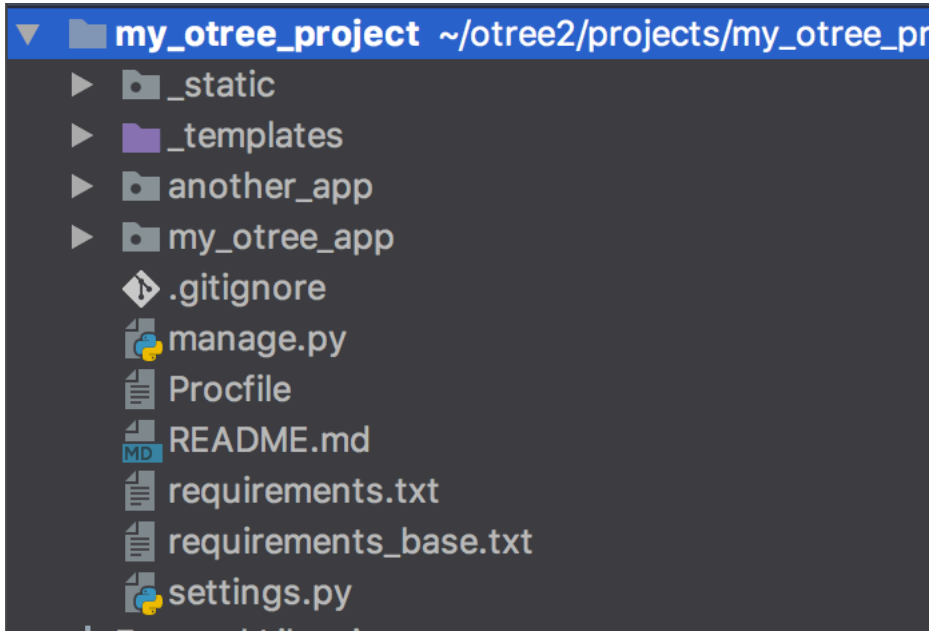
- Templates are just "ordinary" html files which get the info from **`pages`** and show it to a participant.
- As soon as a participant clicks ‘Next’ the data he enters is passed back to **`pages`** which pass them further to **`models`**



One session can contain **several** apps



One session can contain **several** apps

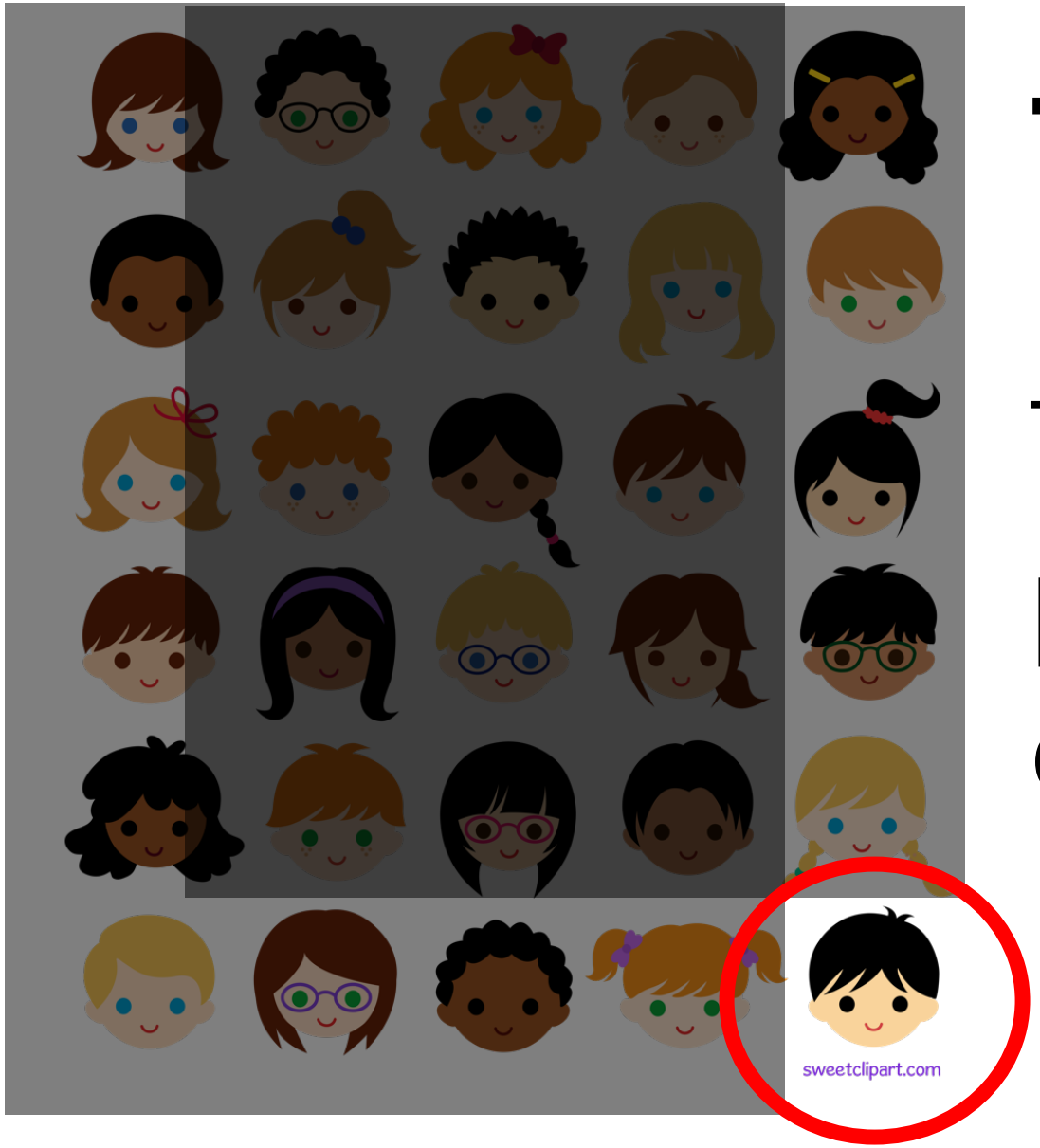


in `settings.py`:

```
SESSION_CONFIGS = [
    {
        'name': 'some_apps',
        'display_name': "Demo apps",
        'num_demo_participants': 1,
        'app_sequence': ['my_otree_app', 'another_app'],
    },
]
```

This is session.

The entire set of all participants in your lab or online



This is participant.

The entire set of all participants in your lab or online

Subsession is a set of all players in one round

Player is an element of subsession

One participant contains the info about all players who he/she ‘owns’

The group
is a set of players
in one particular
subsession

Anatomy of `models.py`

- **`Constants:`** game parameters that stay the same for all players
- **`Subsession:`** parameters that stay the same for all participants in specific round
- **`Group:`** data/parameters that stay the same for all members of one group in specific round
- **`Player:`** data/parameters that are unique for each player in a specific round

Anatomy of an oTree `Page`

- Page creation:
  - class definition
  - template
- When/if/for whom it is shown:
  - position in **`page_sequence`**
  - **`is_displayed`** method
- What is shown:
  - **`vars_for_template`** method
- What to do next:
  - **`before_next_page`** method

Methods of an oTree `Page/WaitPage`

- BEFORE page is shown:
  - `is_displayed` *(should return* **`True`** *if page is to be shown)*
  - `vars_for_template`
- AFTER page is shown:
  - `before_next_page`
- WAITPAGE:
  - `after_all_players_arrive` method

Typical blueprint of a game

1. How many players?
2. Do they interact in the real time?
3. How many rounds?
4. Do players have different roles?
5. What are the main stages within each round?
6. When and how payoffs are calculated?
7. What kind of information a player needs from other players?
8. What kind of external (preexisting) information is needed?
9. How treatments change the game flow?

First step: information about gender

1. Creating app (pgg?)
2. `models.py`: Model field (gender?)
3. `pages.py`: new page class (Gender?)
4. `pages.py`: add class to page_sequence
5. Templates folder: pgg/Gender.html
6. …
7. DONE!

`models.py` – field definition

```
class Constants(BaseConstants):
    GENDERCHOICES = ['Male', 'Female']



class Player(BasePlayer):
    gender = models.StringField(choices=Constants.GENDERCHOICES,
                                label='What is your gender?',
                                widget=widgets.RadioSelectHorizontal)
```

Anatomy of an oTree `Page`

```
class Gender(Page):
    form_model = 'player'
    form_fields = ['gender']

    def is_displayed(self):
        return self.round_number == 1

    def before_next_page(self):
        self.participant.vars['gender'] = self.player.gender


page_sequence = [
    Gender,
]
```

WHY?

Gender.html template

```
{% extends "global/Page.html" %}
{% load otree %}

{% block title %}
    Gender
{% endblock %}

{% block content %}

    {% formfields %}

    {% next_button %}

{% endblock %}
```

Templates: using variables

- In a template you can access variables defined in `vars_for_template` of the specific page:

```
{{ var }}
```

You can also use lists and dictionaries

- You can also access any variable in Constants, Player, Group…

```
{{ Constants.players_per_group }}
{{ player.payoff }}
```

Templates: loops, conditions

- You can use conditions and loops in templates:

```
{% if player.age < 30 %}
    hi, dude!
{% else %}
    Dear Mr. Player,
{% endif %}
```

Templates

- to include another file (for example instructions):

```
{% include ‘path_to_file/name_of_file.html %}
```

- to make ‘Next button’:

```
{% next_button %}
```

Static files: images, css, javascript

in Static folder create a folder with your appname


```
<img src="{% static 'pggfg/pgg.jpg' %}" class="img-fluid" alt="PGG Scheme">
```

Templates

```
{% block content %}

    <div class="card">
        <div class="card-body">
            In this study, you will be in a randomly formed group
            of <b>{{ Constants.players_per_group }}</b> participants.
            You will face a decision situation
            described below. You and other {{ Constants.num_others_per_group }}
            participants like you will
            make this decision for {{ Constants.num_rounds }} rounds.


        </div>
    </div>
    {% include Constants.instructions_template %}
    {% next_button %}

{% endblock %}
```

WHY?

GitHub Education

1. Get GitHub Education!
2. `git init`
3. `git add .`
4. `git commit -m 'Commit message'`
5. `git add remote origin <LINK_TO_GIT>`
6. `git push origin master`
7. rolling back: `git checkout <COMMIT_ID> .`
    1. `git stash`
8. `git pull origin master`

More info: http://rogerdudler.github.io/git-guide/

Heroku

1. create git first!
2. `heroku create <APP_NAME>`
3. `heroku addons:create heroku-redis`
4. `git push heroku master`
5. `heroku run otree resetdb`
6. `heroku config`

7. RECOMMENDED: oTree Hub (www.otreehub.com)

Posting experiments on mTurk

```
AWS_ACCESS_KEY_ID = environ.get('AWS_ACCESS_KEY_ID')
AWS_SECRET_ACCESS_KEY = environ.get('AWS_SECRET_ACCESS_KEY')

mturk_hit_settings = {
    'keywords': ['study', 'experiment', 'academic'],
    'title': 'Academic study',
    'description': 'Behavioral study',
    'frame_height': 1000,
    'preview_template': 'global/MTurkPreview.html',
    'minutes_allotted_per_assignment': 60,
    'expiration_hours': 4,  # 7 days

    'qualification_requirements': [
        {
            'QualificationTypeId': "00000000000000000071",
            'Comparator': "EqualTo",
            'LocaleValues': [{
                'Country': "US",
            }]
        },
    ],

}
```

Creating new app

- open terminal
- change directory to my_first_project folder
- type: otree startapp pgg
- Open the PyCharm
- add the pgg into settings.py:

```
SESSION_CONFIGS = [
    {
        'name': 'pgg',
        'display_name': "Public Good Game",
        'num_demo_participants': 3,
        'app_sequence': ['pgg'],
    },
]
```

Which pages do we need?

Gender: get gender info

Intro: the instructions are shown. We will show it only **once,** at the first round

AfterGenderWP: people wait for Gender answers from others

Contribution: where they will make their decisions on contributions to the common pool

BeforeResultsWP: to calculate payoffs

Results: where they can see the results of a round

FinalResults: the overall results across all n rounds are shown. We will show it only **once** at the final round.

Which pages do we need?

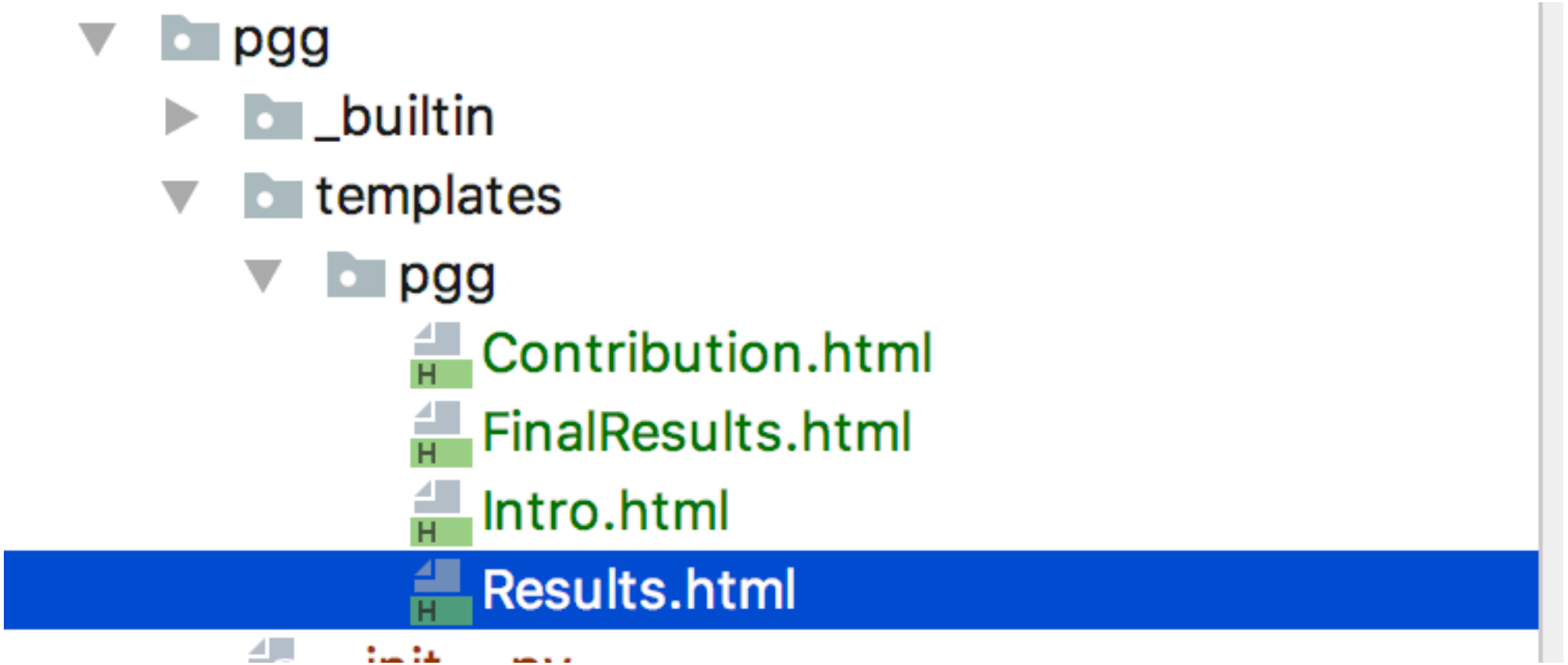
```
class Gender(Page):
    pass

class Intro(Page):
    pass


class AfterGenderWP(WaitPage):
    pass


class Contribution(Page):
    pass


class BeforeResultsWP(WaitPage):
    pass


class Results(Page):
    pass


class FinalResults(Page):
    pass
```

page_sequence in pages.py

```
page_sequence = [
    Gender,
    Intro,
    AfterGenderWP,
    Contribute,
    AfterContribWP,
    Results,
    FinalResults,
]
```

Templates

We need to create corresponding templates:



Conditions on showing pages

```
class Intro(Page):
    def is_displayed(self):
        return self.round_number == 1


...


class FinalResults(Page):
    def is_displayed(self):
        return self.round_number == Constants.num_rounds
```

Constants

```
class Constants(BaseConstants):
    name_in_url = 'pgg'
    players_per_group = 3
    num_others_per_group = players_per_group - 1
    num_rounds = 2
    endowment = c(20)
    efficiency_factor = 1.6
    GENDERCHOICES = ['Male', 'Female']
```

Fields

- Each **player** has his/her initial ***endowment***.
- Each **player** has his/her own ***contribution***.
- The ***sum of contributions*** for the entire **group**
- the ***share of total pie*** that will be distributed to each player in a **group**

Fields

```
class Group(BaseGroup):
    total_contribution = models.IntegerField()
    average_contribution = models.FloatField()
    individual_share = models.CurrencyField()

    def set_payoffs(self):...


class Player(BasePlayer):
    endowment = models.CurrencyField()
    gender = models.StringField(choices=Constants.GENDERCHOICES,
                                label='What is your gender?',
                                widget=widgets.RadioSelectHorizontal)
    contribution = models.PositiveIntegerField()
```

Generating fixed endowments

```
class Subsession(BaseSubsession):
    def creating_session(self):
        for p in self.get_players():
            if self.round_number == 1:
                p.endowment = Constants.endowment
            else:
                p.endowment = p.in_round(1).endowment
```

WHY?

Contribution decision

- **in pages.py:**

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

- in template Contribution.html:

```
<h4>How much you would like to invest to a common group project:</h4>
{% formfield player.contribution label='' %}
```

How do we calculate payoff?

```
class Group(BaseGroup):
    total_contribution = models.IntegerField()
    average_contribution = models.FloatField()
    individual_share = models.CurrencyField()

    def set_payoffs(self):
        self.total_contribution = sum([p.contribution for p in self.get_players()])
        self.average_contribution = round(self.total_contribution / Constants.players_per_group, 2)
        self.individual_share = self.total_contribution * Constants.efficiency_factor / Constants.players_per_group
        for p in self.get_players():
            p.payoff = sum([+ p.endowment,
                            - p.contribution,
                            + self.individual_share,
                            ])
```

Payoff calculation line by line

```
self.total_contribution = sum([p.contribution for p in self.get_players()])
```

- **self.get_players() –** is a method of a Group, that returns the **list** of all players in this group
- we can loop through it obtaining or assigning values for each player in a group
- so this line returns us a list of all contributions of players in a group
- we assign a **sum** of these contributions to a group variable *total_contribution*

Payoff calculation line by line

```
self.individual_share = self.total_contribution * Constants.efficiency_factor/
Constants.players_per_group
```

- we just calculated a total contribution of an entire group. Now we need to calculate how much **each** individual will get from the group project
- this amount is the **same** for each player in a group. So we do it in a group level
- we take a total contribution; we multiply it by multiplication factor; we divide it by total number of players in a group.

Payoff calculation line by line

```
for p in self.get_players():
    p.payoff = sum([+ p.endowment,
                    - p.contribution,
                    + self.individual_share,
                    ])
```

- we loop through a list of all players in our group.
- and for each player we calculate his payoff
- it is:
  - the initial endowment
  - minus his/her contribution
  - plus the individual share from the pie

Calling payoff function

```
class BeforeResultsWP(WaitPage):
    def after_all_players_arrive(self):
        self.group.set_payoffs()
```

- After you have create a function in a group that calculates payoffs, you need to actually call it somewhere.
- It makes sense to do it when all players have made their decisions about contribution
- So in `after_all_players_arrive` function in a Waiting Page

Obtaining data from other players

```
self.group.get_players()
self.player.get_others_in_group()
self.group.get_player_by_role('ROLE_NAME')
self.group.get_player_by_id('ID')


total_contribution = sum([p.contribution for p
in self.get_players()])
```

Obtaining data from previous rounds

- `participant.get_players()`
- `participant.payoff`

- `player.in_all_rounds()`
- `player.in_previous_rounds()`
- `player.in_round(x)`
- `player.in_rounds(x,y)`

- on template:
- `{{player.round_number}}`
- on page:
- `self.round_number`

Results

```
<table class="table">
    <tr>
        <td>Your endowment:</td>
        <td>{{ Constants.endowment }}</td>
    </tr>
    <tr>
        <td>You contributed:</td>
        <td>{{ player.contribution }}</td>
    </tr>
    <tr>
        <td>Your earnings from the project:</td>
        <td><b>{{ group.individual_share }}</b></td>
    </tr>
    <tr>
        <td>Your payoff in this round:</td>
        <td><b>{{ player.payoff }}</b></td>
    </tr>
    <tr>
        <th>Your cumulated payoff so far:</th>
        <th><b>{{ participant.payoff }}</b></th>
    </tr>
</table>
```

Results: other players

- Special method: self.player.get_others_in_group()

```
<h5>Other participants:</h5>
<table class="table table-striped table-hover">
    <thead>
    <th>Participant</th>
    <th>Endowment</th>
    <th>Contribution</th>
    </thead>
    {% for p in player.get_others_in_group %}
        <tr>
            <td> {{ p.id_in_group }}</td>
            <td>{{ p.endowment }}</td>
            <td>{{ p.contribution }}</td>
        </tr>
    {% endfor %}
</table>
```

Results: showing previous rounds

Your cumulated payoff so far:	=	27 points
Your endowment:		**20 points**
You contributed:	-	**2**
Your earnings from the project:	+	**9 points**
Your payoff in this round:	=	**27 points**

Other participants:

Participant	Endowment	Contribution
1	20 points	12
3	20 points	3

Next

Average group contributions across rounds

```
{% if player.round_number > 1 %}
    <h5>Previous periods:</h5>
    <table class="table table-striped table-hover">
        <thead>
        <th>Round</th>
        <th>Your group average contribution</th>
        <th>Your contribution</th>
        </thead>
        <tbody>
        {% for p in player.in_previous_rounds %}
            <tr>
                <td>{{ p.round_number }}</td>
                <td>{{ p.group.average_contribution }}</td>
                <td>{{ p.contribution }}</td>
            </tr>
        {% endfor %}
        </tbody>
    </table>
{% endif %}
```

Results page:

Previous periods:

Round	Your group average contribution	Your contribution
1	13.33	26

Final results:

- We need to show the payoffs of a particular player in each round.

- And the total payoff for the entire game

Final results:

```
<div class="alert alert-danger" role="alert">
    <h5>Your final payoff is: <strong>{{ participant.payoff }}</strong></h5>
</div>

<h5>All periods:</h5>
<table class="table table-striped table-hover">
    <thead>
    <th>Round</th>
    <th>Your group average contribution</th>
    <th>Your contribution</th>
    </thead>
    <tbody>
    {% for p in player.in_all_rounds %}
        <tr>
            <td>{{ p.round_number }}</td>
            <td>{{ p.group.average_contribution }}</td>
            <td>{{ p.contribution }}</td>
        </tr>
    {% endfor %}
    </tbody>
</table>
```

Final results:

Final results

Your earnings:

Round	Average group contribution	Payoff
1	29	116 points
2	22	122 points
Your final payoff:		238 points

Next

Treatments

- Create two different treatments of Public Good game:
- 1. **Baseline** treatment
- 2. **Endowment_heterogeneity**: treatment with heterogeneous endowments (each player has his/her own randomly generated endowment).
- 3. **Gender_info**:Treatment where gender info of other participants is shown
- 4. Gender_info **X** Endowment_heterogeneity treatment

Updated constants:

```
class Constants(BaseConstants):
    name_in_url = 'pgg'
    players_per_group = 3
    num_others_per_group = players_per_group - 1
    num_rounds = 2
    instructions_template = 'pggfg/Instructions.html'
    endowment = c(20)
    endowment_lb = c(10)
    endowment_ub = c(30)
    efficiency_factor = 1.6
    GENDERCHOICES = ['Male', 'Female']
```

Configurating session

```
SESSION_CONFIGS = [
    {
        'name': 'pggfg_baseline',
        'display_name': 'Public Good Game Baseline',
        'num_demo_participants': 3,
        'app_sequence': ['pggfg'],
        'gender': False,
        'hetero': False,
    },
    {
        'name': 'pggfg_gender',
        'display_name': 'Public Good Game - Gender info',
        'num_demo_participants': 3,
        'app_sequence': ['pggfg'],
        'gender': True,
        'hetero': False,
    },
```

Generating treatment info

```
class Subsession(BaseSubsession):
    gender = models.BooleanField()
    hetero = models.BooleanField()

    def creating_session(self):
        self.gender = self.session.config.get('gender')
        self.hetero = self.session.config.get('hetero')

        for p in self.get_players():
            if self.round_number == 1:
                if self.hetero:
                    p.endowment = c(random.randint(Constants.endowment_lb,
                                                   Constants.endowment_ub))
                else:
                    p.endowment = Constants.endowment
            else:
                p.endowment = p.in_round(1).endowment
```

Making instructions flexible

```
In the beginning of each round every participant in your group
of {{ Constants.players_per_group }} receives
{% if subsession.hetero %} a random amount of money from
    {{ Constants.endowment_lb }} to {{ Constants.endowment_ub }}
{% else %}
    <strong>{{ Constants.endowment }}</strong>
{% endif %}. The group
has the opportunity to undertake a joint project. Each participant in
the group decides how much she or he is going to contribute to
```

In the beginning of each round every participant in your group of 3 receives a random amount of money from 10 points to 30 points . The group has the opportunity to undertake a joint project. Each participant in the group decides how much she or he is going to contribute to the project.

In the beginning of each round every participant in your group of 3 receives **20 points** . The group has the opportunity to undertake a joint project. Each participant in the group decides how much she or he is going to contribute to the project.

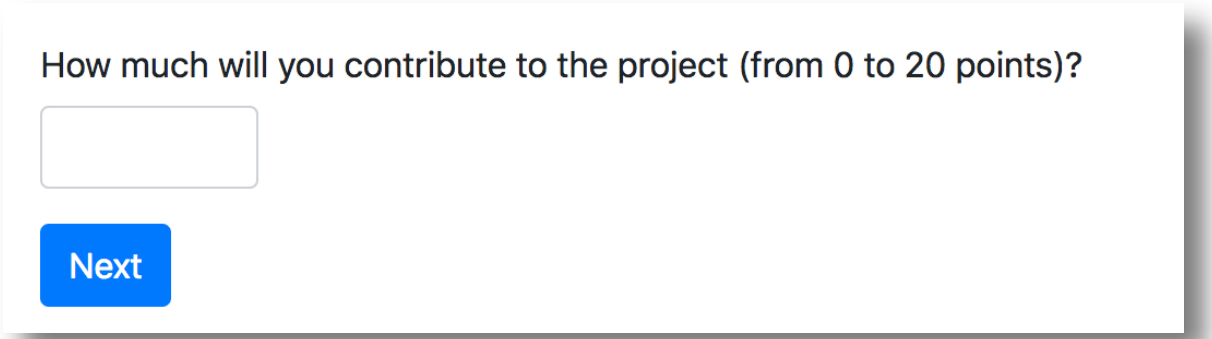
Dynamic min/max

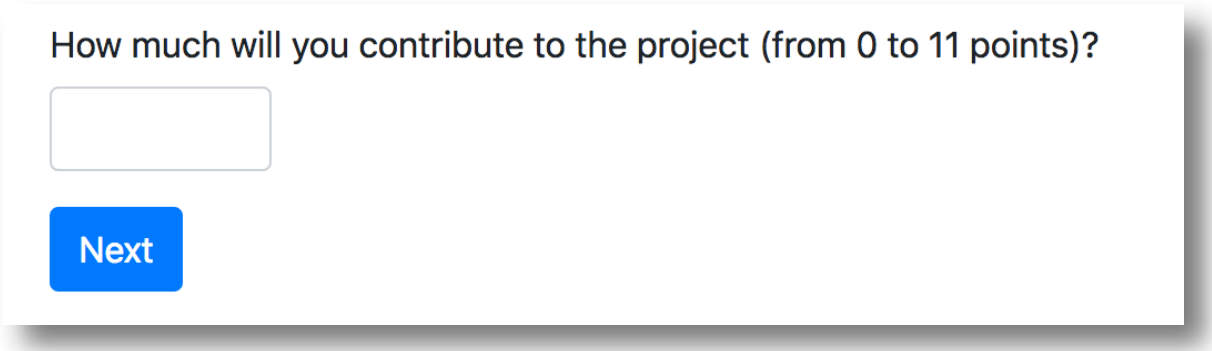
```
class Contribute(Page):
    form_model = 'player'
    form_fields = ['contribution']
    def contribution_max(self):
        return self.player.endowment

    def vars_for_template(self):
        return {'label':"How much will you contribute to the project (from 0 to {})?".
            format(self.player.endowment)}
```

```
{% formfield player.contribution label=label %}
```

Resulting data

	pgg					
	Player					
ID in session	id in group	role	endowment	contribution	payoff	su
P1	1		65		0	
P2	2		61		0	
P3	3		52		0	





Showing gender info on Contribution

```
<table class="table table-striped">
    <thead>
    <th>Participant</th>
    {% if subsession.gender %}
        <th>Gender</th>
    {% endif %}
    <th>Endowment</th>
    </thead>
    <tbody>
    {% for i in player.get_others_in_group %}
        <tr>
            <td> {{ i.id_in_group }}</td>
            {% if subsession.gender %}
                <td>{{ i.participant.vars.gender
            {% endif %}
            <td> {{ i.endowment }}</td>
        </tr>
    {% endfor %}
    </tbody>
</table>
```

Participant	Gender	Endowment
1	Female	20 points
3	Male	20 points

Participant	Endowment
2	26 points
3	29 points

Gender info on Results page

Other participants:

Participant	Gender	Endowment	Contribution
1	Female	10 points	2
3	Male	26 points	10

Next

Gender info on Results page

```
<table class="table table-striped table-hover">
    <thead>
    <th>Participant</th>
    {% if subsession.gender %}
        <th>Gender</th>{% endif %}
    <th>Endowment</th>
    <th>Contribution</th>
    </thead>
    {% for p in player.get_others_in_group %}
        <tr>
            <td> {{ p.id_in_group }}</td>
            {% if subsession.gender %}
                <td>{{ p.participant.vars.gender }}</td>{% endif %}
            <td>{{ p.endowment }}</td>
            <td>{{ p.contribution }}</td>
        </tr>
    {% endfor %}
</table>
```

timeouts

```
class MyPage(Page):
    form_model = ‘player’
    form_fields = [‘accept’]
    timeout_seconds = 600
    timeout_submission = {'accept': True}
    def before_next_page(self):
     if self.timeout_happened:
            self.player.accept = random.choice([True, False])
```

shuffling players

- Subsession:
  - self.group_randomly()
  - self.group_randomly(fixed_id_in_group=True)
  - self.group_like_round()

Rooms:

```
ROOMS = [{
    'name': 'zurich01',
    'display_name': 'Room for Zurich workshop',

}]
```

Room: Room for Zurich workshop

Create a new session

Session config:

Number of participants

Create

0 participants present

Persistent URLs

These URLs will stay constant for new sessions, even if the database is

Room-wide URL

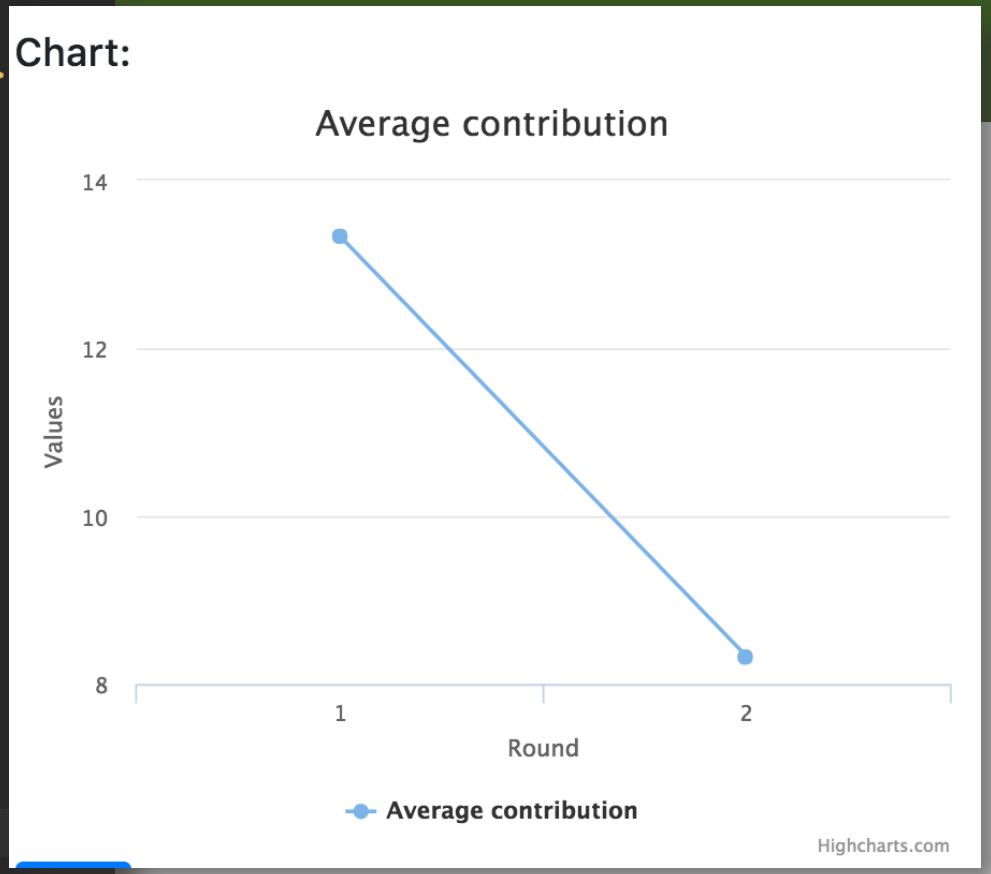
Here is the room-wide URL anyone can use. Don't use this link when tes
tabs will be assigned to the same participant, using a cookie.

https://zurich-workshop-2018.herokuapp.com/room/zurich01/

Charts

```
{% block scripts %}
    <script src="//code.highcharts.com/highcharts.js"></script>

    <script>
        $(function () {
            Highcharts.chart('highcharts-container', {
                title: {
                    text: 'Average contribution'
                },
                xAxis: {
                    categories: {{ round_numbers|json }},
                    title: {text: 'Round'}
                },
                series: [{{ highcharts_series|json }}]
            });
        });
    </script>
{% endblock %}
```

Chart:




```
class Results(Page):
    def vars_for_template(self):
        return {'round_numbers': [g.round_number for g in self.group.in_all_rounds()],
                'highcharts_series': {'name': 'Average contribution',
                                      'data': [g.average_contribution for g in self.group.in_all_rounds()]}}
```

oTree extensions and add-ons

- otreeutils
- otree-tools
- oTree Hub (www.otreehub.com)
- oTree-manager:
  - Demo: http://demo.otree-manager.com
  - Documentation http://docs.otree-manager.com
  - Code: https://github.com/chkgk/otree_manager
- oTree Virtual Machine Manager:
  - Documentation: https://otree-virtual-machine-manager.readthedocs.io/

Dynamic set of forms

- in Page class:
- class MyPage(Page):

```
def get_form_fields(self):
    if self.player.treatment == ‘A’:
            return [‘a’, ‘b’]
    else:
             return [‘c’, ‘d’]
```

Dynamic set of choices

- you have a field *fruit*
- in Page class:

```
import random
class MyPage(Page):
def fruit_choices(self):
    choices = ['apple', 'kiwi', 'mango']
    random.shuffle(choices)
    return choices
```

Randomizing treatments

```
class Subsession(BaseSubsession):
    def creating_session(self):

        if self.round_number == 1:
            for p in self.get_players():
                p.treatment = random.choice(['A','B']

            for g in self.get_groups()
                g.treatment = random.choice(['A','B']

        else:

            for p in self.get_players():
                p.treatment = p.in_round(1).treatment
```

Roles

```
class Player(BasePlayer):
    def role(self):
        if self.id_in_group==1:
            return 'dictator’
        else:
            return 'receiver’
```

- `role()` is a method of a Player’s model
- we can use id of a player within a group to assign roles
- …. or any other logic

Conditional view based on role

```
class Decision(Page):
    form_model=group'
    form_fields = ['dg_decision']

    def is_displayed(self):
        return self.player.role() == 'dictator'
```

- `is_displayed` is a method built-in in a Page
- by default it is shown to all players
- if under certain condition it returns `False`, it is skipped

Question randomization

```
class Constants(BaseConstants):
    qs_a = ['political_views', 'trust']
    qs_b = ['experimenter_demand', 'gender']
    question_sets = [qs_a, qs_b]

class Subsession(BaseSubsession):
    def creating_session(self):
        if self.round_number == Constants.num_rounds:
            for p in self.get_players():
                qs = Constants.question_sets.copy()
                for i in qs:
                    random.shuffle(i)
                random.shuffle(qs)
                p.qs_order = json.dumps(qs)
```

Question randomization

```
class Survey1(SurveyPage):
    def get_form_fields(self):
        return json.loads(self.player.qs_order)[0]

class Survey2(SurveyPage):
    def get_form_fields(self):
        return json.loads(self.player.qs_order)[1]
```

	Player		
role	qs order	political views	tru
sender	[["experim enter_dem and", "gend...		
receiver	[["trust", "political_v iews"],...		