Programming and conducting experiments

Università degli Studi di Torino
Dipartimento di Culture, Politica e Società

Matteo Ploner (UNITN)

05 November, 2020

Fundamental components of an experiment

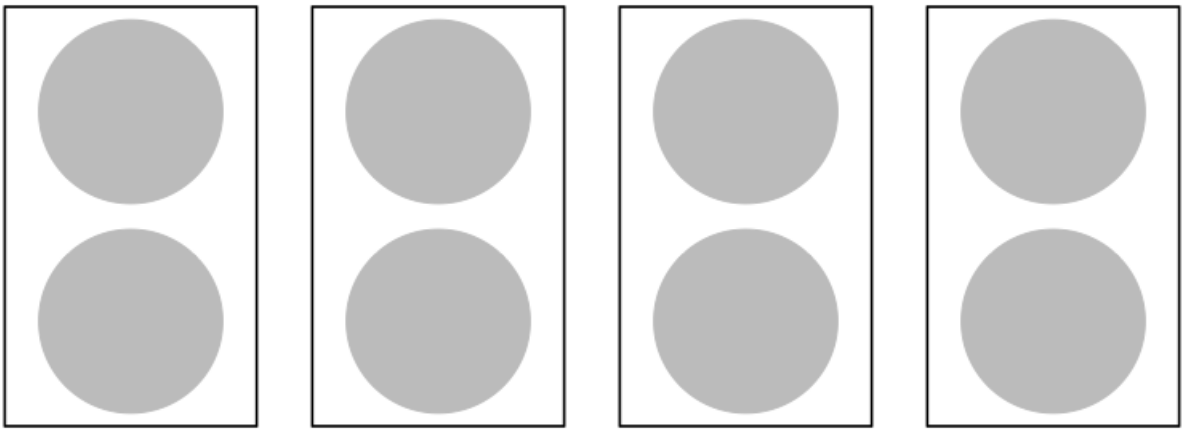
1 Design of the experiment

1.1 Matching, roles, values

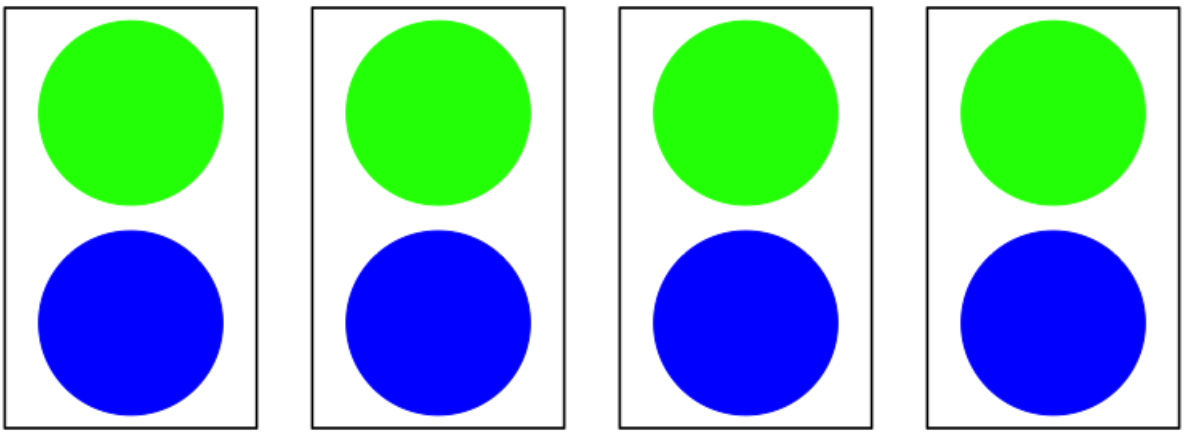
- How many **repetitions** of an interaction are implemented?
  - One shot
  - Repeated
- When repeated, how are people **matched**?
  - Partner matching
  - Random partner matching
- Which **roles** do they have in the interaction?
  - Symmetric roles
  - Asymmetric roles
- Which **values** are associated to sujects?
  - Unconditional
  - Conditional

1.2 A working example

- Consider 8 subjects that are matched in groups of two



- One subject in each group is of type BLUE and the other of type GREEN



1.3 Repeated interaction

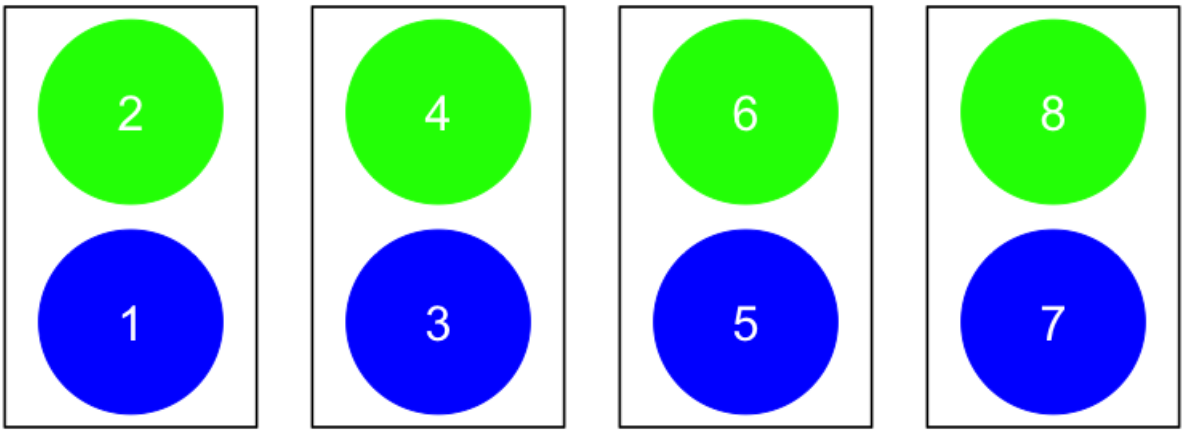
- When the interaction is repeated we can have 4 possible combinations of type ad matching

		Matching	
		Constant	*Varying*
Type	*Constant*	CT/CM	CT/VM
	Varying	VT/CM	VT/VM

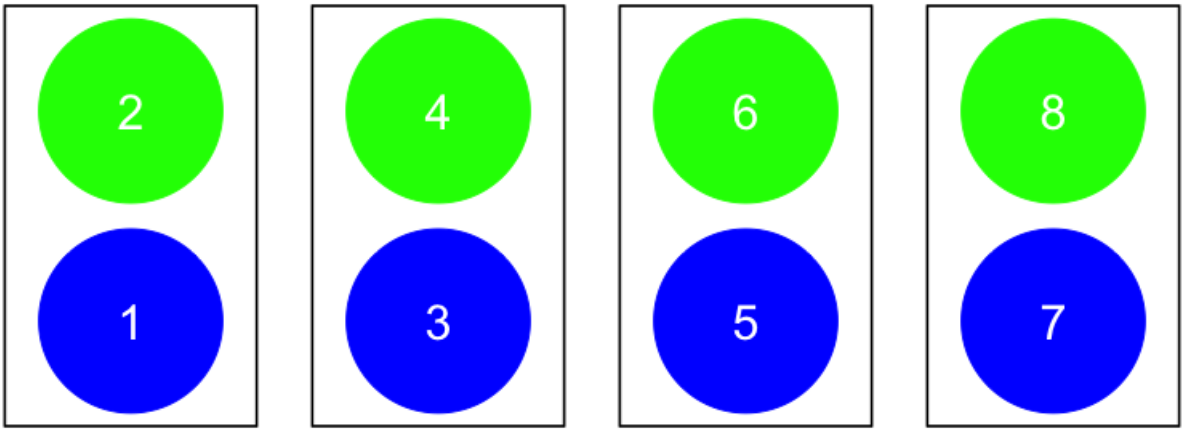
- Changes in roles and matching can be explicitly modeled by design or be random
  - In the code provided below we *randomize*
    - Safe and common approach

1.4 Constant Type and Constant Matching (CT/CM)

- Round 1



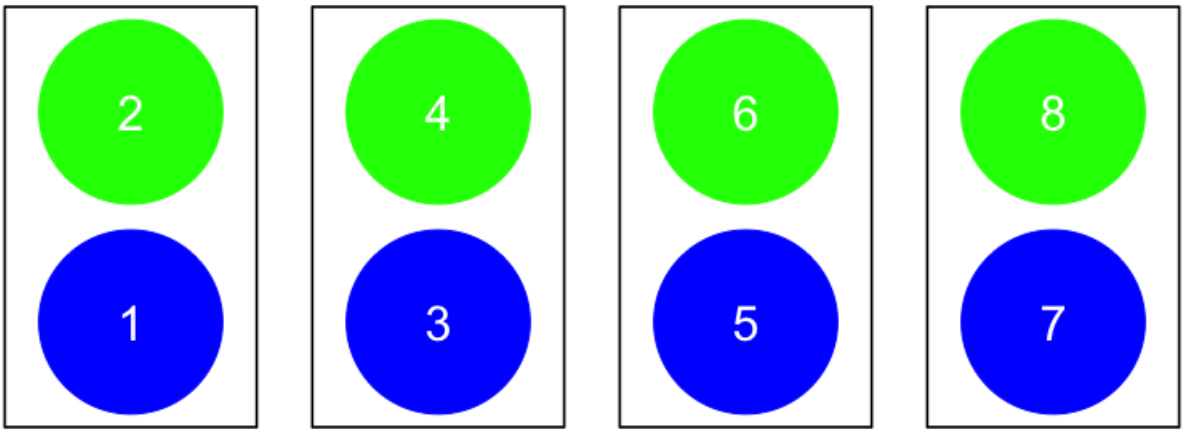
- Round 2



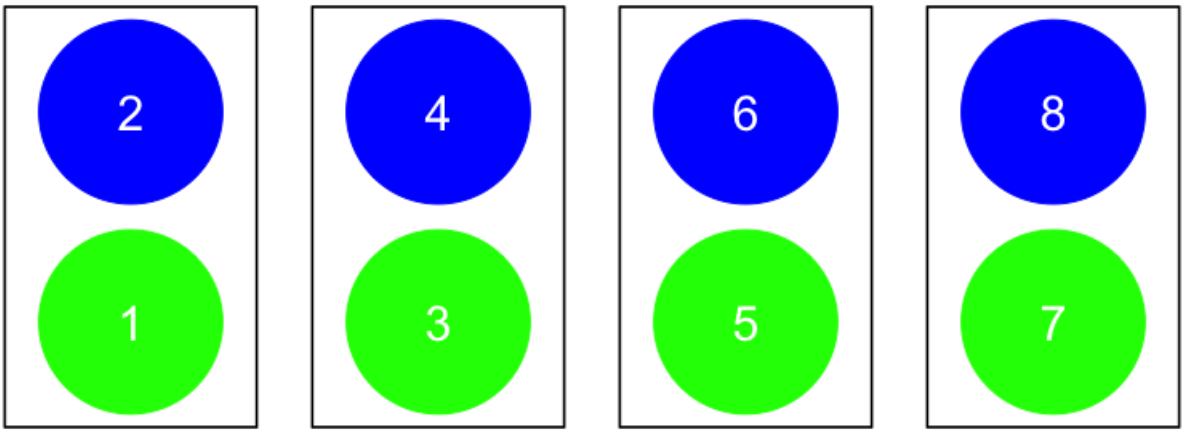
- ...
- Individuals are matched together for the entire experiment and keep the same role across repetitions

1.5 Varying Type and Constant Matching (VT/CM)

- Round 1



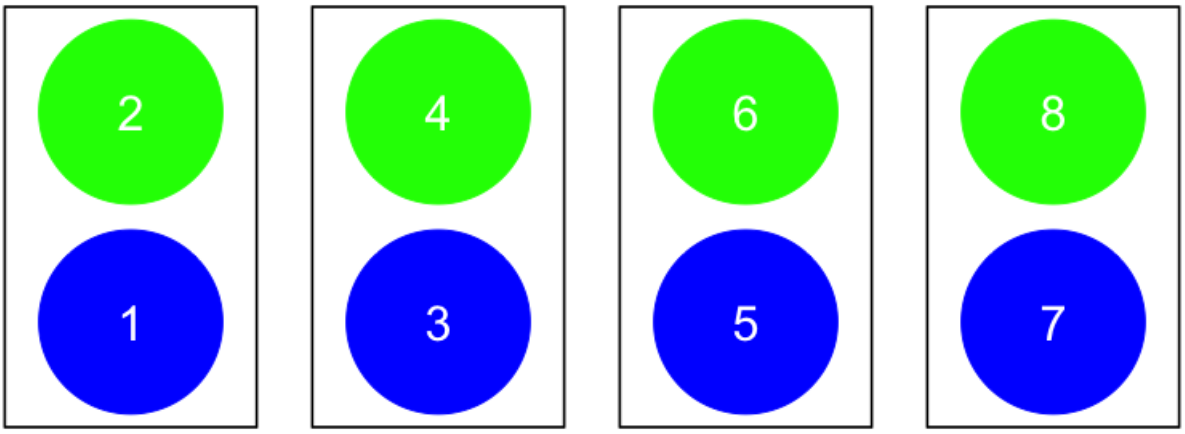
- Round 2



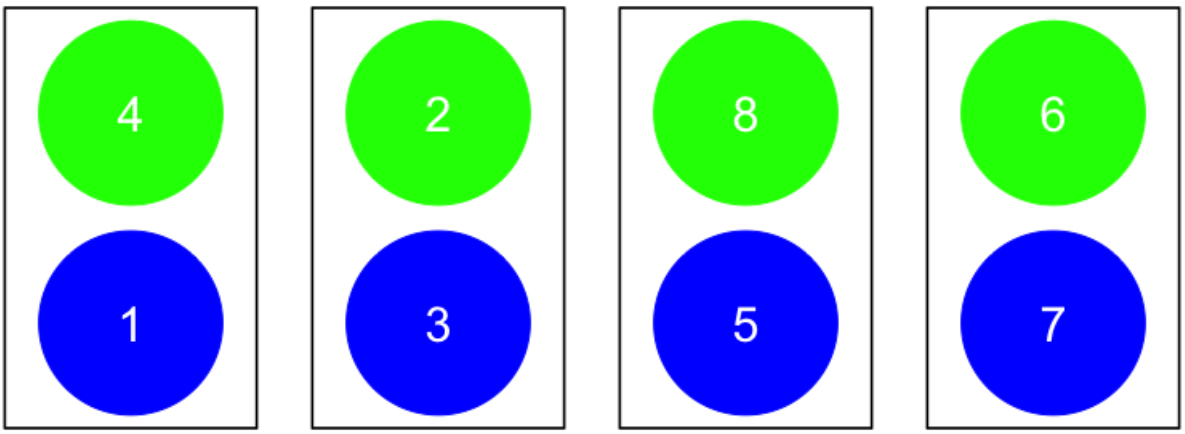
- ...
- Individuals are matched together for the entire experiment but do not keep the same role across repetitions

1.6 Constant Type and Varying Matching (CT/VM)

- Round 1



- Round 2

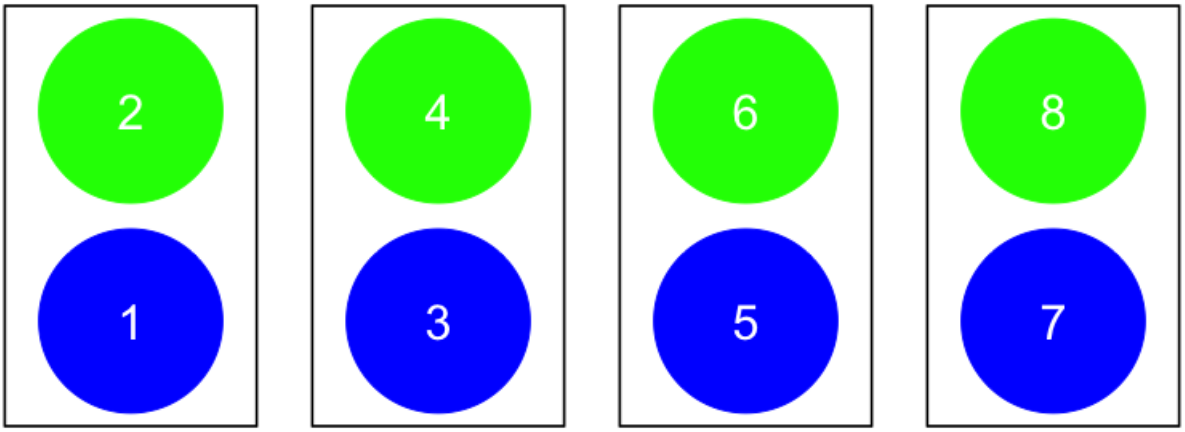


- ...

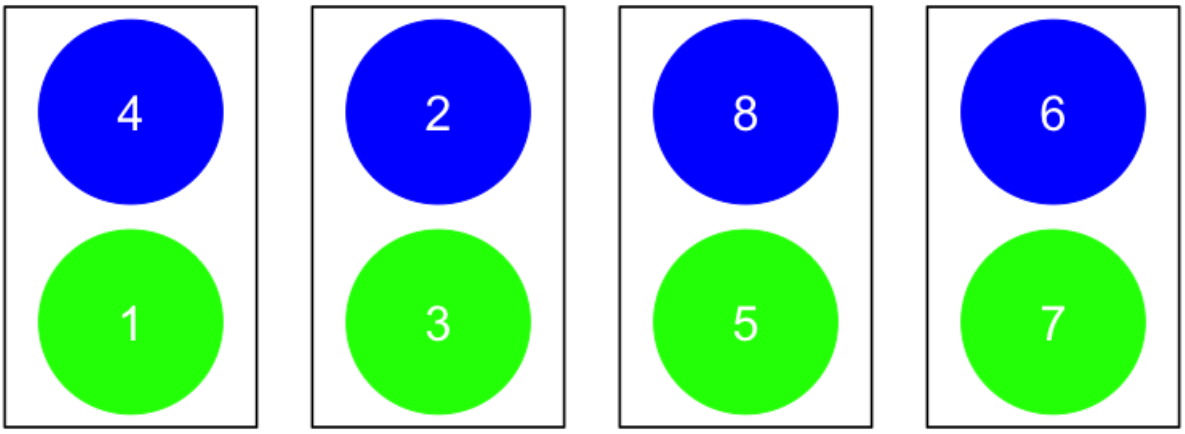
- Individuals are not matched together for the entire experiment but keep the same role across repetitions

1.7 Varying Type and Varying Matching (VT/VM)

- Round 1



- Round 2



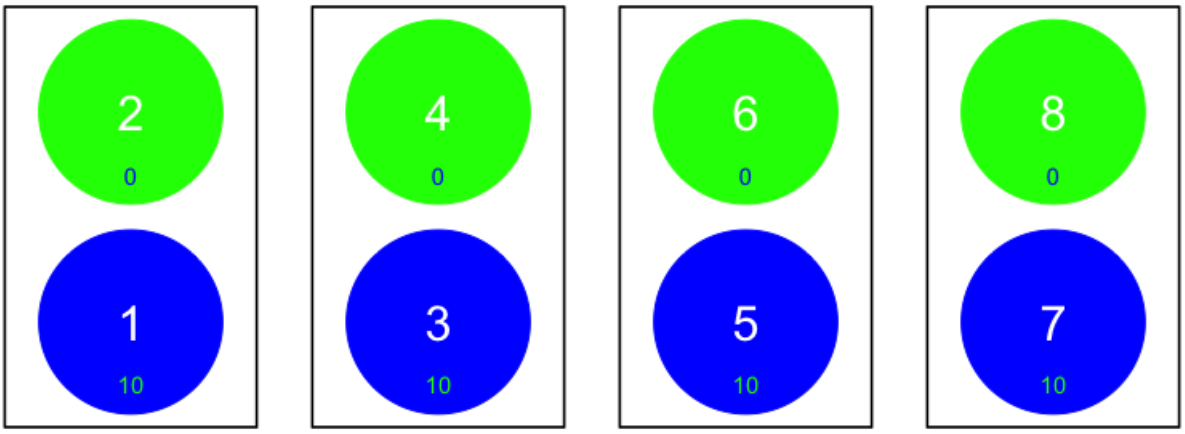
- ...
- Individuals are not matched together for the entire experiment and do not keep the same role across repetitions

1.8 Values

- Participants are generally endowed with some values (attributes)
  - Unconditionally the same for all participants
    - e.g., endowment in the dictator game
  - Conditional upon some characteristic of the participant
    - e.g., efficiency factors the are related to previous performance in a task

1.9 Values: example

- Green Players get an endowment of 0 Euro
- Blue players get an endowment of 10 Euro



2 oTree code for roles, matching, and value assignment

2.1 models.py

- Roles and matching are governed by the file **models.py**
  - This file manages the “structure” of your experiment
    - The “database”
- 4 alternative protocols are presented here
  - Constant type and constant matching (CT/CM)
  - Varying type and constant matching (VT/CM)
  - Constant type and varying matching (CT/VM)
  - Varying type and varying matching (VT/VM)

2.2 Constant type and constant matching (CT/CM)

- This is the code in *models.py*

```
class Constants(BaseConstants):
    name_in_url = 'groups_roles'
    players_per_group = 2
    num_rounds = 10
    matching = "Constant Type and Constant Matching (CT/CM)"

class Subsession(BaseSubsession):
    def creating_session(self):
        if self.round_number == 1: # this way we get a fixed role across repetitic
            self.group_randomly()
            print(self.get_group_matrix())
            for g in self.get_groups():
                for p in g.get_players():
                    if p.id_in_group % 2 == 0:
```

```
                        p.type = 'BLUE'
                        p.value = c(10) # assign the corresponding value
                    else:
                        p.type = 'GREEN'
                        p.value = c(0)# assign the corresponding value
        else:
            self.group_like_round(1)
            for g in self.get_groups():
                for p in g.get_players():
                    p.type = p.in_round(self.round_number-1).type

class Group(BaseGroup):
    pass

# needed to store values

class Player(BasePlayer):
    type = models.StringField()
    id_oth = models.IntegerField()
    type_oth = models.StringField()
    value = models.CurrencyField()
    value_oth = models.CurrencyField()
```

- This is the assignment to groups and roles we get (from the POW of Player 1)

Round: 1

Constant Type and Constant Matching (CT/CM)

	ID	Type	Value
Self	1		10 points
Other	9		0 points

Next

Round: 2

Constant Type and Constant Matching (CT/CM)

	ID	Type	Value
Self	1		10 points
Other	9		0 points

Next

...

Round: 10

Constant Type and Constant Matching (CT/CM)

	ID	Type	Value
Self	1		10 points
Other	9		0 points

Next

2.3 CT/CM: commented code

```
class Constants(BaseConstants):
# define here the constants of the session
    name_in_url = 'groups_roles'
```

```
    # label that appears in browser
    players_per_group = 2
    # how many players in each group (important for matching!)
    num_rounds = 10
    # how many repetitions (important for matching!)
    matching = "Constant Type and Constant Matching (CT/CM)"
    # name of the matching protocol

class Subsession(BaseSubsession):
#group and types are defined in the Subsession class
    def creating_session(self):
    #to set initial values in the subsession
#*****************************************************************************
# START code to generate matching and roles in Round 1
#*****************************************************************************
        if self.round_number == 1:
        # the following code is executed only id round is == 1
        # round_number -> is a built-in function that gives the current round numb
            self.group_randomly()
            # group_randomly() -> built-in function that  shuffles players randoml
            for g in self.get_groups():
            # get_groups() -> returns a list of all the groups in the subsession.
            # loop through the groups in the subsession
                for p in g.get_players():
                # get_players() -> returns a list of all the players in the subses
                # loop through the players in the subsession (p is a player)
                    if p.id_in_group % 2 == 0:
                    # id_in_group -> player's attribute (unique identifier)
                    # if the id is even (via modulo operator)
                        p.type = 'BLUE'
                        # the participant is assigned to type "BLUE"
                        # type is "initialized" in class player as a string
                        p.value = c(10)
                        # the blues are assigned an endowment of 10 points
                        # value is "initialized" in class player as currency
                    else:
                      # if the participant id is odd
                        p.participant.vars['type'] = 'GREEN'
                        # the participant is assigned to type "GREEN"
                        p.value = c(10)
                        # the greens are assigned an endowment of 0 points
                        # value is "initialized" in class player as currency

#*****************************************************************************
# END code to generate matching and roles in Round 1
#*****************************************************************************
#*****************************************************************************
# START code to generate matching and types in round >1
#*****************************************************************************

        else:
          # if round is not round 1 (see the indenting)
            self.group_like_round(1)
```

```
            # perform matching like in round 1 (partner matching)
            for g in self.get_groups():
                for p in g.get_players():
                    p.type = p.in_round(self.round_number-1).type
                    p.value= p.in_round(self.round_number-1).value
#*****************************************************************************
# END code to generate matching and roles
#*****************************************************************************


class Player(BasePlayer):
    type = models.StringField() # this is a string variable that will be filled wi
    value = models.CurrencyField() # this is a currency variable that will be fill

class Group(BaseGroup):
    pass
```

2.4 Varying type and constant matching (VT/CM)

```
class Constants(BaseConstants):
    name_in_url = 'groups_roles'
    players_per_group = 2
    num_rounds = 10
    matching = " Varying Type and Constant Matching (VT/CM)"

import random
class Subsession(BaseSubsession):
    def creating_session(self):
        if self.round_number == 1: # this way we get a fixed role across repetitic
            self.group_randomly()# built-in function
            print(self.get_group_matrix())
            rdm=random.randint(0, 1)
            print(rdm)
            for g in self.get_groups():
                for p in g.get_players():
                    if rdm==1: #this way we randomize role accordin to id in grou
                        if p.id_in_group % 2 == 0:
                            p.type = 'BLUE'
                            p.value = c(10)# assign the corresponding value
                        else:
                            p.type = 'GREEN'
                            p.value = c(0)   # assign the corresponding value
                    else:
                        if p.id_in_group % 2 == 0:
                            p.type = 'GREEN'
                            p.value = c(10)
                        else:
                            p.type = 'BLUE'
                            p.value = c(0)
```

```
        else:
            self.group_like_round(1)
            rdm=random.randint(0, 1)
            print(rdm)
            for g in self.get_groups():
                for p in g.get_players():
                    if rdm==1: #this way we randomize role accordin to id in grou
                        if p.id_in_group % 2 == 0:
                            p.type = 'BLUE'
                            p.value = c(10)
                        else:
                            p.type = 'GREEN'
                            p.value = c(0)
                    else:
                        if p.id_in_group % 2 == 0:
                            p.type = 'GREEN'
                            p.value = c(10)
                        else:
                            p.type = 'BLUE'
                            p.value = c(0)

class Player(BasePlayer):
    type = models.StringField() # this is a string variable that will be filled wi
    value = models.CurrencyField() # this is a currency variable that will be fill

class Group(BaseGroup):
    pass
```

- This is the assignment to groups and roles we get (from the POW of Player 1)

Round: 1

Varying Type and Constant Matching (VT/CM)

	ID	Type	Value
Self	1		10 points
Other	5		0 points

Next

...

Round: 3

Varying Type and Constant Matching (VT/CM)

	ID	Type	Value
Self	1		0 points
Other	5		10 points

Next

...

Round: 10

Varying Type and Constant Matching (VT/CM)

	ID	Type	Value
Self	1		0 points
Other	5		10 points

Next

2.5 VT/CM: commented code

```
class Constants(BaseConstants):
# define here the constants of the session
    name_in_url = 'groups_roles'
    # label that appears in browser
    players_per_group = 2
    # how many players in each group (important for matching!)
    num_rounds = 10
    # how many repetitions (important for matching!)
    matching = " Varying Type and Constant Matching (VT/CM)"
    # matching protocol

import random
# import module random
class Subsession(BaseSubsession):
#group and types are defined in the Subsession class
    def creating_session(self):
    #to set initial values in the subsession
#*****************************************************************************
# START code to generate matching and types in round 1
#*****************************************************************************
        if self.round_number == 1:
        # the following code is executed only id round is == 1
        # round_number -> is a built-in function that gives the current round numb
            self.group_randomly()# built-in function
            # group_randomly() -> built-in function that  shuffles players randoml
            rdm=random.randint(0, 1)
            # assign a random value, either 0 or 1, to variable rdm
            for g in self.get_groups():
            #get_groups() -> returns a list of all the groups in the subsession.
             # loop through the groups in the subsession
                for p in g.get_players():
                # get_players() -> returns a list of all the players in the subses
                # loop through the players in the subsession (p is a player)
#*****************************************************************************
# matching and types when random is 1 - > even = BLUE, odd= GREEN
#*****************************************************************************
                    if rdm==1:
                    # the following code is executed if rdm is 1
                        if p.id_in_group % 2 == 0:
                        # id_in_group -> player's attribute (unique identifier)
                        # if the id is even (via modulo operator)
                            p.type = 'BLUE'
                            # the participant is assigned to type "BLUE"
                            p.value = c(10)# assign the corresponding value
                            # the participant is assigned the corresponding endowm
                          else:
                          # if the participant id is odd
                            p.type = 'GREEN'
                            # the participant is assigned to type "GREEN"
                            p.value = c(0)# assign the corresponding value
```

```
                            # the participant is assigned the corresponding endowm

#*****************************************************************************
# matching and types when random is 1 - > even = GREEN, odd= BLUE
#*****************************************************************************
                    else:
                    # the following code is executed if rdm is 0
                          if p.id_in_group % 2 == 0:
                          # see comment above
                              p.type = 'GREEN'
                              # see comment above
                              p.value = c(0)# assign the corresponding value
                            # the participant is assigned the corresponding endowm
                          else:
                              p.type = 'BLUE'
                              # see comment above
                              p.value = c(10)# assign the corresponding value
                            # the participant is assigned the corresponding endowm
#*****************************************************************************
# END code to generate matching and types in round 1
#*****************************************************************************
#*****************************************************************************
# START code to generate matching and types in round 1
#*****************************************************************************
        else:
        # if round is not round 1 (see the indenting)
            self.group_like_round(1)
            # perform matching like in round 1 (partner matching)
            rdm=random.randint(0, 1)
            # here we run the same code as in round 1 to randomly generate types
            for g in self.get_groups():
                for p in g.get_players():
                    if rdm==1:
                        if p.id_in_group % 2 == 0:
                            p.type = 'BLUE'
                        else:
                            p.type = 'GREEN'
                    else:
                        if p.id_in_group % 2 == 0:
                            p.type = 'GREEN'
                        else:
                            p.type = 'BLUE'
#*****************************************************************************
# END code to generate matching and types
#*****************************************************************************

class Group(BaseGroup):
    pass

# needed to store values

class Player(BasePlayer):
```

```
    type = models.StringField()
    value = models.CurrencyField()
```

2.6 Constant Type and Varying Matching (CT/VM)

```
class Constants(BaseConstants):
    name_in_url = 'groups_roles'
    players_per_group = 2
    num_rounds = 2
    matching = "Constant Type and Varying Matching (CT/VM)"

class Subsession(BaseSubsession):
    def creating_session(self):
        self.group_randomly(fixed_id_in_group=True)
        print(self.get_group_matrix())
        for g in self.get_groups():
            for p in g.get_players():
                if p.id_in_group % 2 == 0:
                    p.type = 'BLUE'
                    p.value = c(10)
                else:
                    p.type = 'GREEN'
                    p.value = c(0)

class Player(BasePlayer):
    type = models.StringField()

class Group(BaseGroup):
    pass
```

- This is the assignment to groups and roles we get (from the POW of Player 1)

Round: 1

Constant Type and Varying Matching (CT/VM)

	ID	Type	Value
Self	1		0 points
Other	10		10 points

Next

...

Round: 3

Constant Type and Varying Matching (CT/VM)

	ID	Type	Value
Self	1		0 points
Other	6		10 points

Next

...

Round: 10

Constant Type and Varying Matching (CT/VM)

	ID	Type	Value
Self	1		0 points
Other	4		10 points

Next

2.7 CT/VM: commented code

```
class Constants(BaseConstants):
# define here the constants of the session
    name_in_url = 'groups_roles'
    # label that appears in browser
    players_per_group = 2
    # how many players in each group (important for matching!)
    num_rounds = 10
    # how many repetitions (important for matching!)
    matching = "Varying Type and Varying Matching (VT/VM)"
    # name of matching protocol

class Subsession(BaseSubsession):
#group and types are defined in the Subsession class
    def creating_session(self):
    #to set initial values in the subsession
        self.group_randomly(fixed_id_in_group=True)
        #group_randomly(fixed_id_in_group=True) -> built-in function that  shuffle
        for g in self.get_groups():
        # get_groups() -> returns a list of all the groups in the subsession.
        # loop through the groups in the subsession
            for p in g.get_players():
            # get_players() -> returns a list of all the players in the subsessior
            # loop through the players in the subsession (p is a player)
                if p.id_in_group % 2 == 0:
                  # id_in_group -> player's attribute (unique identifier)
                  # if the id is even (via modulo operator)
                    p.type = 'BLUE'
                    # the participant is assigned to type "BLUE"
                    p.value = c(10)
                    # the corresponding endowment
                else:
                # if the participant id is odd
                    p.type = 'GREEN'
                    # the participant is assigned to type "GREEN"
                    p.value = c(0)
                    # the corresponding endowment

class Player(BasePlayer):
    type = models.StringField()
    value = models.CurrencyField()
```

```
class Group(BaseGroup):
    pass
```

2.8 Varying Type and Varying Matching

```
class Constants(BaseConstants):
    name_in_url = 'groups_roles'
    players_per_group = 2
    num_rounds = 2

import random
class Subsession(BaseSubsession):
    def creating_session(self):
        self.group_randomly()#
        rdm=random.randint(0, 1)
        for g in self.get_groups():
            for p in g.get_players():
                if rdm==1:
                    if p.id_in_group % 2 == 0:
                        p.type = 'BLUE'
                    else:
                        p.type = 'GREEN'
                else:
                    if p.id_in_group % 2 == 0:
                        p.type = 'GREEN'
                    else:
                        p.type = 'BLUE'

class Player(BasePlayer):
    type = models.StringField()
    value = models.CurrencyField()

class Group(BaseGroup):
    pass
```

Round: 1

Varying Type and Varying Matching (VT/VM)

	ID	Type	Value
Self	1		10 points
Other	9		0 points

Next

Round: 2

Varying Type and Varying Matching (VT/VM)

	ID	Type	Value
Self	1		0 points
Other	9		10 points

Next

...

Round: 10

Varying Type and Varying Matching (VT/VM)

	ID	Type	Value
Self	1		0 points
Other	8		10 points

Next

2.9 VT/VM: commented code

```
class Constants(BaseConstants):
# define here the constants of the session
    name_in_url = 'groups_roles'
    # label that appears in browser
    players_per_group = 2
    # how many players in each group (important for matching!)
    num_rounds = 2
    # how many repetitions (important for matching!)
    matching = "Varying Type and Varying Matching (VT/VM)"
    # matching protocol

import random
class Subsession(BaseSubsession):
#group and types are defined in the Subsession class
    def creating_session(self):
    #to set initial values in the subsession
        self.group_randomly()
        #group_randomly() -> built-in function that  shuffles players
        rdm=random.randint(0, 1)
        # assign a random value, either 0 or 1, to variable rdm
        for g in self.get_groups():
        #get_groups() -> returns a list of all the groups in the subsession.
        # loop through the groups in the subsession
            for p in g.get_players():
            # get_players() -> returns a list of all the players in the subsessior
            # loop through the players in the subsession (p is a player)
#*************************************************************************
# matching and types when random is 1 - > even = BLUE, odd= GREEN
#*************************************************************************
                if rdm==1:
                # the following code is executed if rdm is 1
                    if p.id_in_group % 2 == 0:
                    # id_in_group -> player's attribute (unique identifier)
                    # if the id is even (via modulo operator)
                        p.type = 'BLUE'
                        # the participant is assigned to type "BLUE"
                        p.value = c(10)
                    else:
                    # if the participant id is odd
                        p.type = 'GREEN'
                        # the participant is assigned to type "GREEN"
```

```
                        p.value = c(0)
#*****************************************************************************
# matching and types when random is 1 - > even = GREEN, odd= BLUE
#*****************************************************************************
                else:
                # the following code is executed if rdm is 0
                    if p.id_in_group % 2 == 0:
                    # see comment above
                        p.type = 'GREEN'
                        # see comment above
                        p.value = c(0)
                    else:
                    # see comment above
                        p.type = 'BLUE'
                        # see comment above
                        p.value = c(10)
#*****************************************************************************
# END code to generate matching and types
#*****************************************************************************

class Group(BaseGroup):
    pass

class Player(BasePlayer):
    type = models.StringField()
    value = models.CurrencyField()
# needed to store values
```

2.10 Thank you

3 Appendix

3.1 Assignments

- Easy
  - Implement a *constant type and constant matching (CT/CM)* with value assignment as follows
    - All types get 10 Euros
      - Green types get additional 2 Euros
      - Blue types are taken away 2 Euros
- Less easy
  - Implement the following matching protocol
    - 8 periods
    - First 4 periods varying type and varying matching (random) (VT/VM)
    - Last 4 periods constant types constant matching (CT/CM)

3.2 OTree code

- The oTree app of this lecture:

Download groups_roles.zip

3.3 References