

A Comprehensive Introduction to z-Tree

Stefan Palan

ztree@palan.biz

<http://academic.palan.biz>



References

- Fischbacher, U., *z-Tree Tutorial*, www.iew.uzh.ch/ztree/ztree21tutorial.pdf, 3rd of January, 2002. (Tutorial)
- Fischbacher, U., *z-Tree - Zurich Toolbox for Readymade Economic Experiments - Experimenter's Manual*, Institute for Empirical Research in Economics, University of Zurich, Working Paper No. 21, ISSN 1424-0459, 1999. (Manual)
- Fischbacher, U., *z-Tree Reference Manual*, <http://www.iew.uzh.ch/ztree/ztree21ref.pdf>, 6th of January, 2006. (Reference Manual)
- Z-Tree Wiki, <https://www.uzh.ch/iew/ztree/ssl-dir/wiki/>, 26th of March, 2012. (Wiki)
- Lecture ressources: academic.palan.biz - select Downloads

Introduction to Experimentation with z-Tree

- Structuring a session
- Characteristics and features of z-Tree
- Networking
- Command line options
- Files used by z-Tree

Structuring a typical session 1/3

1. Subject arrival

- Subjects checked on list
- Excess subjects are sent home with show-up fee
- Other subjects are (randomly) assigned to workstations

2. Instructions on mechanism

- Hand out printed instructions, read them out loud
- Control questions

3. Training round(s)

4. Bathroom break

5. Instructions on parameters

- Hand out printed instructions, read them out loud
- Control questions

6. Run treatment

...Repeat 5 & 6 for multiple parameter constellations...

7. Questionnaires

- Elicit questionnaire responses
- Elicit risk-aversion, etc.

8. Payment

- Random number generation
 - As transparent as possible (e.g. real dice)
 - Random numbers for individual subjects reduce cost variance but increase required time
- Pay subjects individually and anonymously
- Have subjects sign receipt
- Ask subjects not to talk about experiment with others

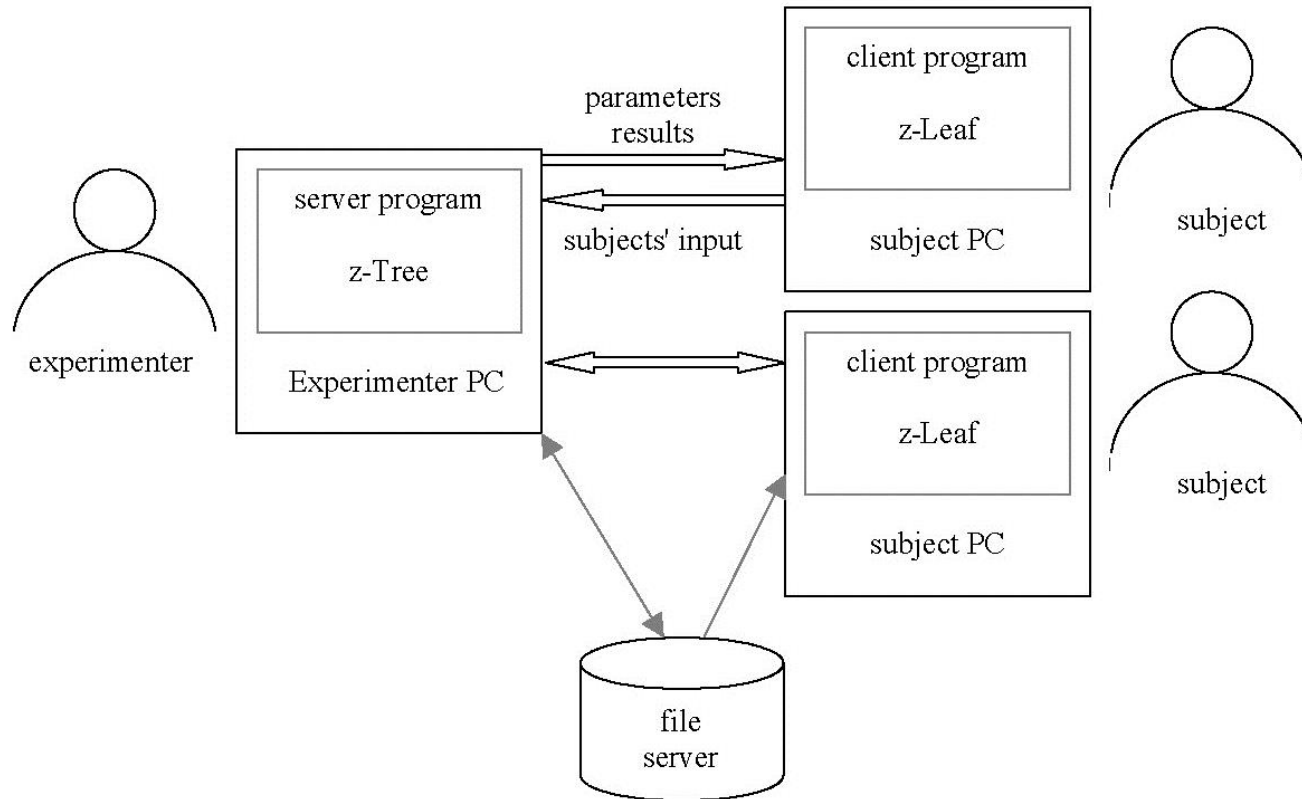
General procedural steps

- Follow session structure cheat sheet diligently
 - Contains procedural steps
 - Contains program settings
 - Contains required material (dice, cards, etc.)
- Note special occurrences in experimenter diary
- Backup everything

Zurich **T**oolbox for **R**eadymade **E**conomic **E**xperiments

- Simple, flexible programming language for economic experiments
- Client-server architecture
- Automatic networking, data collection and payoff calculation
- Crash recovery capability

Network topology



Network connections: Critical issues

- Setup with file server
 - Ensure server has read/write access
 - Ensure client has read access
 - Put all files into one directory
 - Clients can be restarted by *Run-Restart All Clients*
- Setup without file server
 - Ensure server has read/write access
 - Point client to server IP
 - Clients can only be restarted manually

Network connections: Critical issues

- Server IP (in server.eec)
 - Command line parameter, e.g.: `zleaf /server 10.0.0.1`
 - IP file is in the zleaf.exe-directory
 - IP file is in the directory `c:\expecon\conf`
 - IP equals local machine's IP
- Client name
 - Command line parameter, e.g.: `zleaf /name pc1`
 - File name.eec in zleaf.exe-directory
 - Name equals local machine's TCP/IP hostname

Other important command line options

- **zleaf.exe**
 - **Language:** `/language`
e.g. `/language en`, **or** `/language english`
 - **Screen resolution:** `/size`
e.g. `/size 1024x768`
- **ztree.exe**
 - **Change default directories**
 - `/xlmdir`, `/sbjdir`, `/adrdir`, `/paydir`, `/gsfdir`, `/tempdir`,
`/datadir` (autosaves of `.ztt` and `.ztq`), `/leafdir` (`server.eec`)
 - e.g. `/adrdir c:\data`

Files used by z-Tree

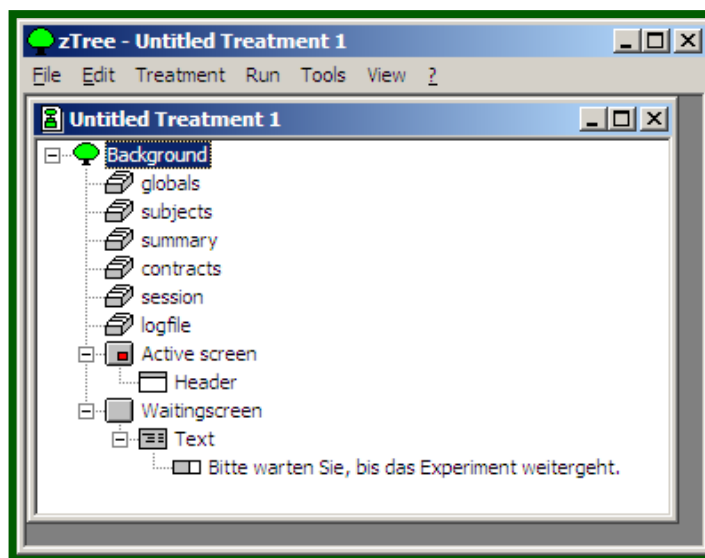
ztree.exe	Server program file
zleaf.exe	Client program file
name.ztt	Treatment code file
name.ztq	Questionnaire code file
name.txt	Parameter input file
@1.ztt, @2.ztt, ...	Backups of treatments and questionnaires
@1.ztq, @2.ztq, ...	
@db.txt, @lastclt.txt, @prevdb.txt	Temporary files
090330_0804.adr	Subject address file
090330_0804.gsf	Crash recovery file (binary)
090330_0804.pay	Payout information file
090330_0804.sbj	Questionnaire response file
090330_0804.xls	Main data file (~ MS Excel compatible)

Public Goods Game Tutorial

- Description of the game
- Programming the game
- Advice for testing
- Resulting output
- Questionnaire

Example: Public goods experiment*

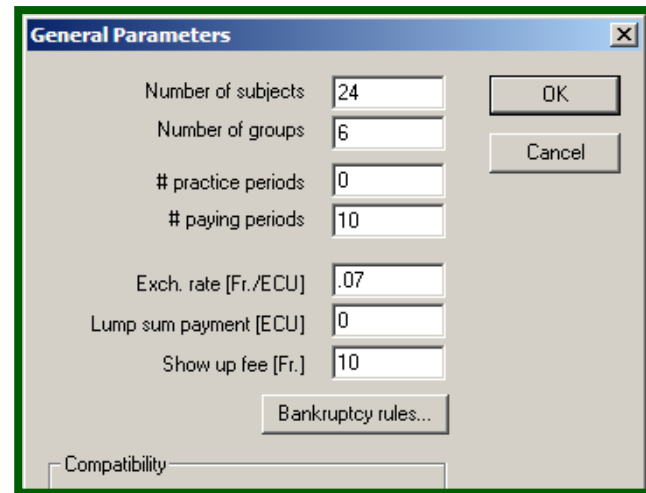
- Groups of $n = 4$ subjects
- Initial endowment of subject i : $w_i = 20$
- Contribution of subject i : $c_i \in [0, w_i]$
- Profit of subject i : $\pi_i = w_i - c_i + 1.6 \cdot \frac{1}{n} \cdot \sum_{j=1}^4 c_j$
- New z-Tree treatment:



* Source: z-Tree Tutorial, sections 2.2-2.3.

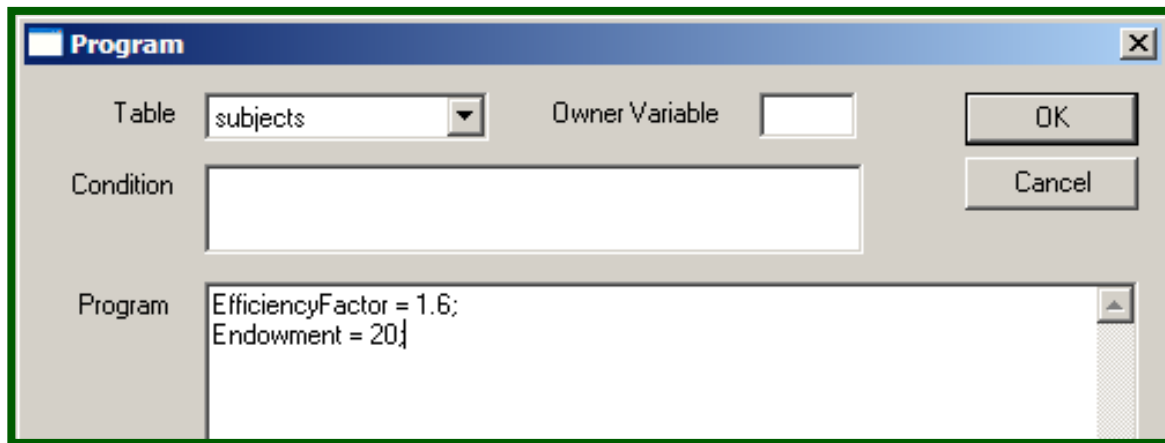
Define parameters

- Define general parameters in Background: 
- Create new program
 - Select logfile in stage tree
 - Click Treatment-New Program...



Parameter	Value
Number of subjects	24
Number of groups	6
# practice periods	0
# paying periods	10
Exch. rate [Fr./ECU]	.07
Lump sum payment [ECU]	0
Show up fee [Fr.]	10

Buttons: OK, Cancel, Bankruptcy rules..., Compatibility

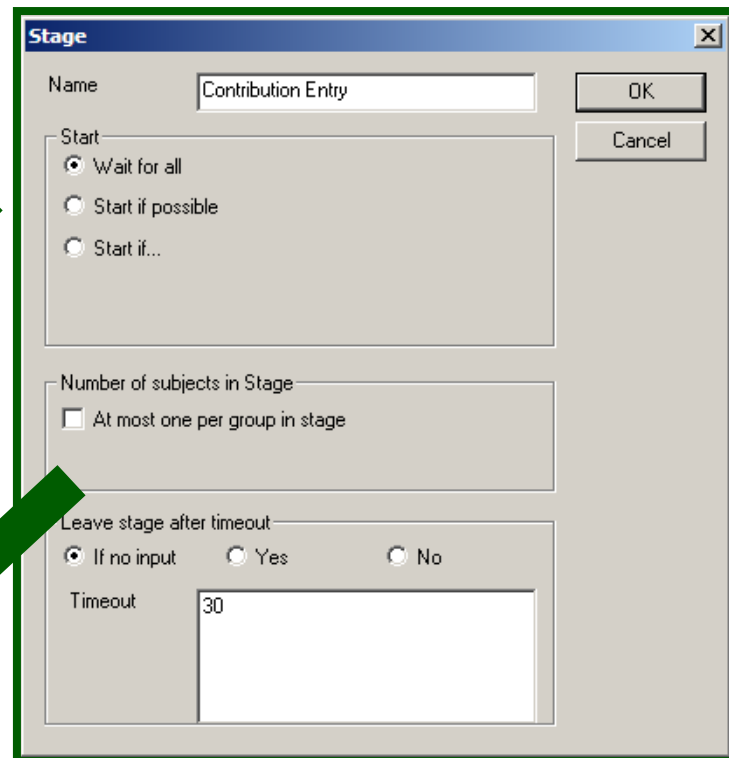
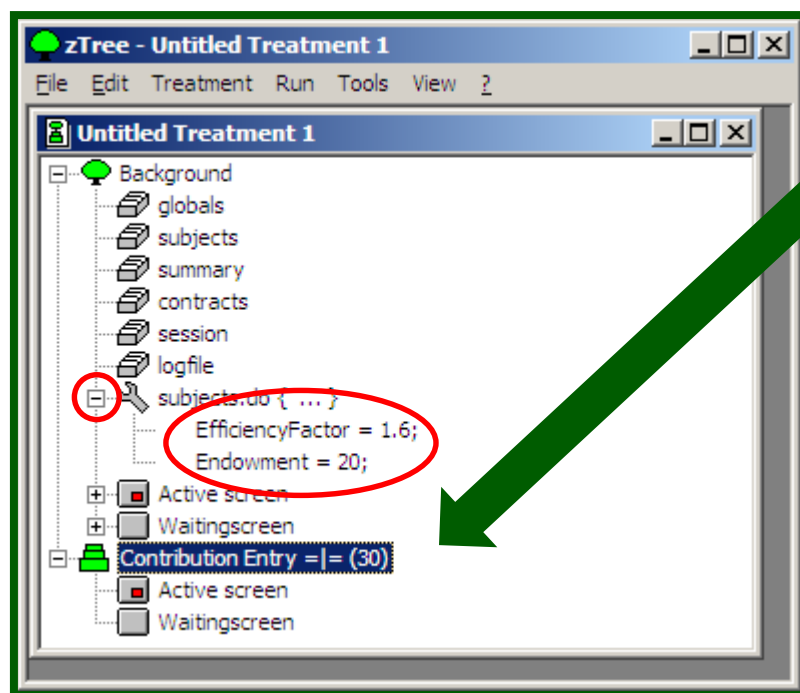


Field	Value
Table	subjects
Owner Variable	
Condition	
Program	EfficiencyFactor = 1.6; Endowment = 20;

Buttons: OK, Cancel

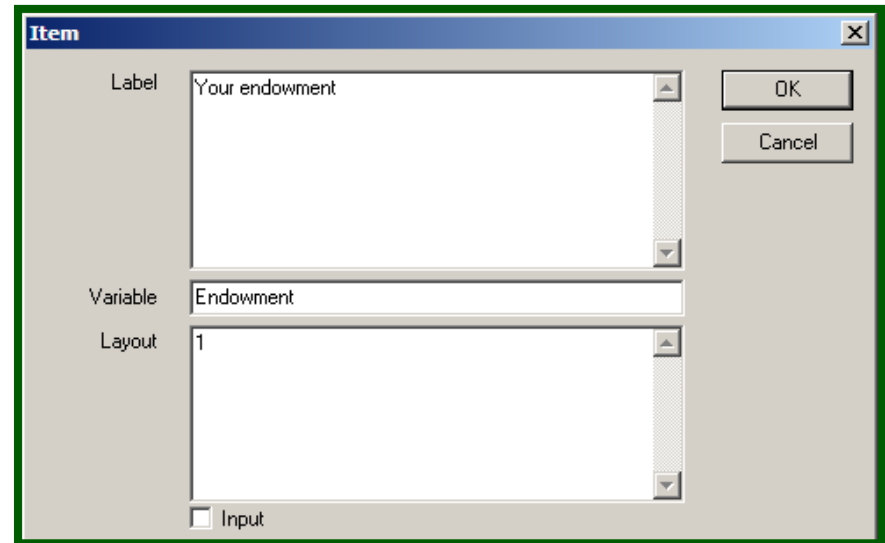
The first stage

- Add the first stage
 - Select the Background
 - Click Treatment-New Stage...



Designing the screen 1/2

- Add the first box
 - Select Active Screen in the Contribution Entry stage
 - Click Treatment-New Box>Standard Box...
 - Click OK
- Add an output item
 - Select the new Standard Box
 - Click Treatment-New Item...



Designing the screen 2/2

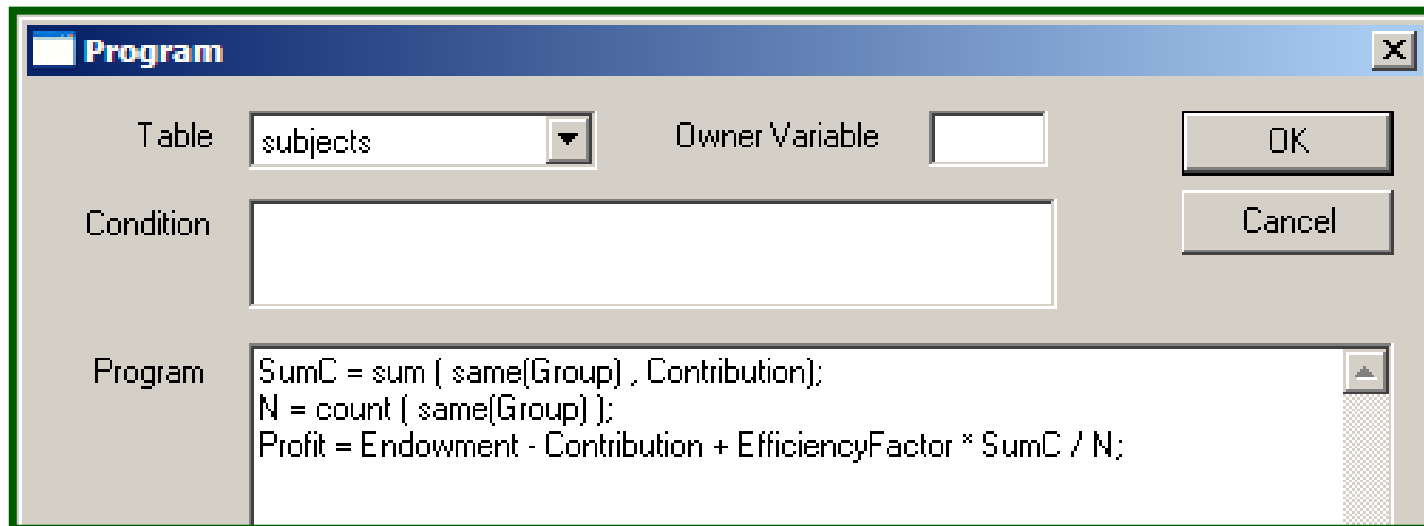
- Add an input item
 - Select the first item
 - Click Treatment-New Item...
- Add a button
 - Select the input item
 - Click Treatment-New Button...
- Enter “OK” as name and click OK

The screenshot shows the 'Item' dialog box in the z-Tree software. The dialog has a title bar with 'Item' and a close button. It contains several fields and checkboxes:

- Label:** A text field containing 'Your contribution to the project'.
- Variable:** A text field containing 'Contribution'.
- Layout:** A text field containing '1'.
- Input:** A checkbox that is checked.
- Minimum:** A text field containing '0'.
- Maximum:** A text field containing 'Endowment'.
- Show value (value of variable or default):** An unchecked checkbox.
- Empty allowed:** An unchecked checkbox.
- Default:** An empty text field.
- Buttons:** 'OK' and 'Cancel' buttons are located in the top right corner.

Profit calculation

- Add new stage Profit Display
- Add profit calculation
 - Select Profit Display stage
 - Click Treatment-New Program...



The screenshot shows a dialog box titled "Program" with a close button (X) in the top right corner. The dialog contains three main input areas:

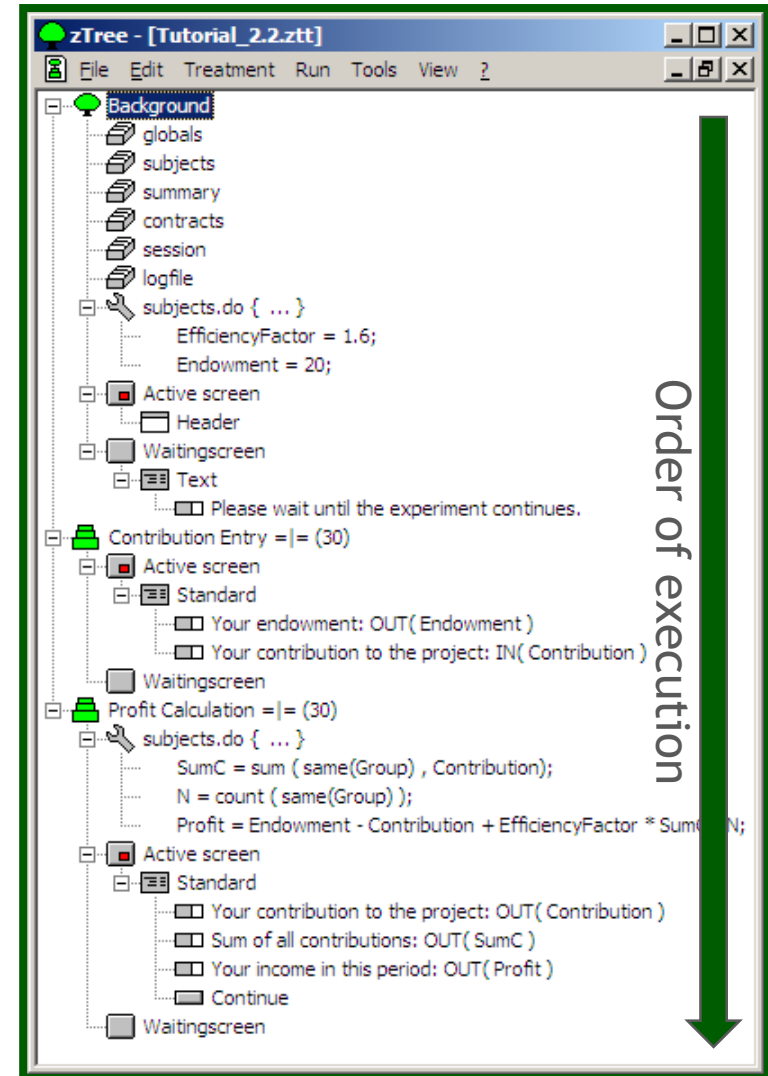
- Table:** A dropdown menu currently showing "subjects".
- Owner Variable:** An empty text input field.
- Condition:** A large empty text area.
- Program:** A text area containing the following code:

```
SumC = sum ( same(Group) , Contribution);  
N = count ( same(Group) );  
Profit = Endowment - Contribution + EfficiencyFactor * SumC / N;
```

On the right side of the dialog, there are two buttons: "OK" and "Cancel".

Profit display

- Add new box to Active screen of Profit Display stage
- Insert display items
 - Own contribution
 - Sum of all contributions
 - Subject's income for the period
- Insert “Continue” button
- Save the treatment (.ztt file)



Link to
Starting multiple z-Leafs for testing
in
Good Practice and Advice

Advice for testing 1/2

- Switch between leafs using <ALT>-<TAB>
- Kill leafs using <ALT>-<F4>
- Display connected clients by clicking Run-Clients Table

The image shows three sequential screenshots of a window titled "Clients' Table". Each window contains a table with 4 columns: "4 clients", "state", and "time". The rows represent four players: Player1, Player2, Player3, and Player4. Green arrows point from the first window to the second, and from the second to the third, indicating a progression of time.

4 clients	state	time
Player1		
Player2		
Player3		
Player4		

4 clients	state	time
Player1	- Contribution Entry -	27
Player2	*** Contribution Entry ***	27
Player3	- Contribution Entry -	27
Player4	*** Contribution Entry ***	27

4 clients	state	time
Player1	Ready	-
Player2	Ready	-
Player3	Ready	-
Player4	Ready	-

- Pause time using <F12>
- Resume time using <Shift>-<F12>
- Stop testing by clicking Run-Stop after this period

Advice for testing 2/2

- To test at your leisure:
- Make periods very short (e.g. 2 sec)
- Start program (<F5>)...
- ...then immediately pause the time (<F12>)
- Test at will
- Resume/Pause to move to next period (<Shift>-<F12>)
- Click Run-Stop after this period to initiate end of testing
- End testing by clicking “OK” on all waiting screens

Resulting screen output

Period 1 of 10 Time remaining [sec]: 27

Period 1 of 10 Time remaining [sec]: 26

Your contribution to the project 7
Sum of all contributions 26
Your income in this period 23.4

subjects table

Period	Subject	Group	Profit	TotalProfit	Participate	EfficiencyFactor	Endowment	Contribution	TimeOK	ContributionEntryOK	SumC	N	TimeContinue	ProfitCalculationOK	CalculationOK
1	1	1	23.4	23.4	1	1.6	20	7	99999	26	4	0			
1	2	1	27.4	27.4	1	1.6	20	3	99999	26	4	-			
1	3	1	19.4	19.4	1	1.6	20	11	99999	26	4	-			
1	4	1	25.4	25.4	1	1.6	20	5	99999	26	4	-			

Resulting data output

- Output in YYMMDD_hhmm.xls:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	090707_0824	1	globals	Period	NumPeriods	RepeatTreatment									
2	090707_0824	1	globals	1	10	0									
3	090707_0824	1	subjects	Period	Subject	Group	Profit	TotalProfit	Participate	EfficiencyFactor	Endowment	Contribution	TimeOKContributionEntryOK	SumC	N Tim
4	090707_0824	1	subjects	1	1	1	19	19	1	1.6	20	13	29	30	4
5	090707_0824	1	subjects	1	2	1	30	30	1	1.6	20	2	26	30	4
6	090707_0824	1	subjects	1	3	1	24	24	1	1.6	20	8	24	30	4
7	090707_0824	1	subjects	1	4	1	25	25	1	1.6	20	7	20	30	4
8	090707_0824	1	globals	Period	NumPeriods	RepeatTreatment									
9	090707_0824	1	globals	2	10	0									
10	090707_0824	1	subjects	Period	Subject	Group	Profit	TotalProfit	Participate	EfficiencyFactor	Endowment	Contribution	TimeOKContributionEntryOK	SumC	N Tim
11	090707_0824	1	subjects	2	1	1	28.4	47.4	1	1.6	20	4	29	31	4
12	090707_0824	1	subjects	2	2	1	25.4	55.4	1	1.6	20	7	27	31	4
13	090707_0824	1	subjects	2	3	1	12.4	36.4	1	1.6	20	20	25	31	4
14	090707_0824	1	subjects	2	4	1	32.4	57.4	1	1.6	20	0	23	31	4
15	090707_0824	1	globals	Period	NumPeriods	RepeatTreatment									
16	090707_0824	1	globals	3	10	0									
17	090707_0824	1	subjects	Period	Subject	Group	Profit	TotalProfit	Participate	EfficiencyFactor	Endowment	Contribution	TimeOKContributionEntryOK	SumC	N Tim
18	090707_0824	1	subjects	3	1	1	19.6	67	1	1.6	20	14	28	34	4
19	090707_0824	1	subjects	3	2	1	26.6	82	1	1.6	20	7	22	34	4
20	090707_0824	1	subjects	3	3	1	25.6	62	1	1.6	20	8	20	34	4
21	090707_0824	1	subjects	3	4	1	28.6	86	1	1.6	20	5	17	34	4
22	090707_0824	1	summary	Period											
23	090707_0824	1	summary	1											
24	090707_0824	1	summary	2											
25	090707_0824	1	summary	3											
26	090707_0824	1	session	Subject	FinalProfit	ShowUpFee	ShowUpFeeInvested	MoneyAdded	MoneyToPay	MoneyEarned					
27	090707_0824	1	session	1	4.69	10	0	0	14.69	14.69					
28	090707_0824	1	session	2	5.74	10	0	0	15.74	15.74					
29	090707_0824	1	session	3	4.34	10	0	0	14.34	14.34					
30	090707_0824	1	session	4	6.02	10	0	0	16.02	16.02					

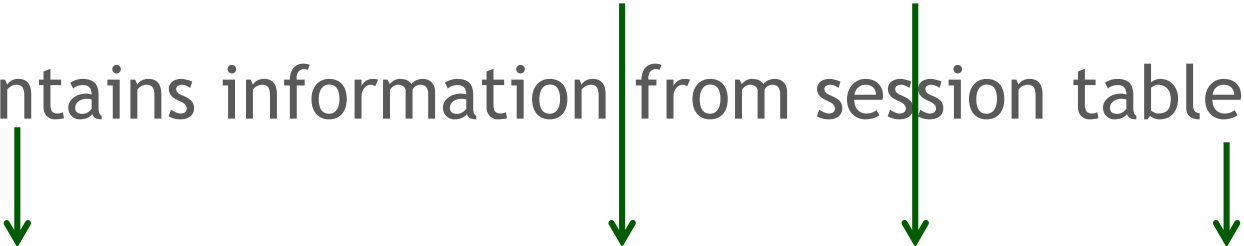
Creating new questionnaire

- Click File-New Questionnaire
- Click Questionnaire-New Address Form
 - Questions left empty will not be asked
 - May be entirely empty
 - Address form plus empty question form suffice to complete questionnaire
 - Run questionnaire (may be empty) at least once to write payment file

The screenshot displays a software window titled "Untitled Questionnaire 1". Inside, there is a tab labeled "Address" which contains a form titled "Adress" (note the spelling). The form consists of several input fields with corresponding labels: "Adress Entry" (containing "Address"), "First Name" (containing "First Name"), "Last Name" (containing "Surname"), "Adress" (containing "Street"), "Postal code" (containing "Postal code"), "City" (containing "Town"), "Telephone" (containing "Telephone"), and "E-Mail" (containing "Email"). To the right of these fields are "OK" and "Cancel" buttons. Below the input fields, there is a question: "Do you want to participate in further experiments?". This is followed by two sets of radio buttons labeled "Yes" and "No". At the bottom of the form, there are two more input fields: "Continue (button label)" containing "continue" and "Help" containing "Help". A "Help text" area at the very bottom contains the text: "Please enter your name and adress. Your entries are confidential." The entire window is framed with a standard operating system border.

Payment file

- Created after treatment and questionnaire have been run
- Contains information from address form
- Contains information from session table



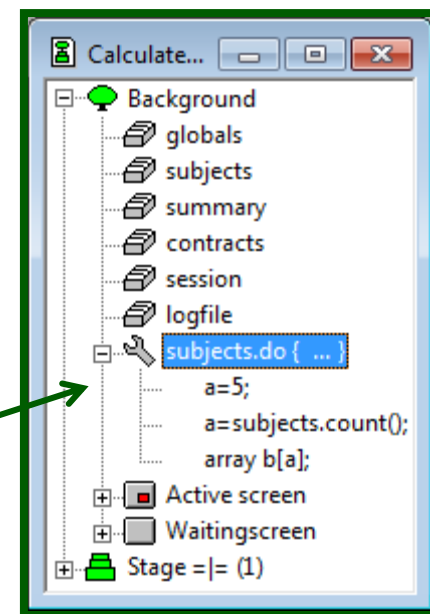
Subject	Computer	Interested	Name	Profit	Signature
1	Player1	OK	Jane Parker	21.52	
2	Player2	OK	Jim Smith	15.43	
3	Player3	OK	John Doe	20.00	
4	Player4	OK	Bill Farmer	17.23	
Experiment	C:\Institut\zTree\120328_1127.pay			74.18	

Basics of z-Tree Programming

- Variable types and simple functions
- Structure of a treatment
 - Background
 - Creating a stage
 - Period structure
 - Parameter table
- Loops

Variable types

- Floating point variables
 - Standard variable type
 - Saved in binary format (minimal deviations from decimal format)
- Array variables
 - Used to save (infrequently) recurring values
 - Array size can be fixed...
 - `array a[3];`
 - `array a[1,5,2];`
 - `a[2]=10;`
 - ... or calculated with a “trick”
- String variables
 - New in Version 3.4.0 and later
 - `b=""`;

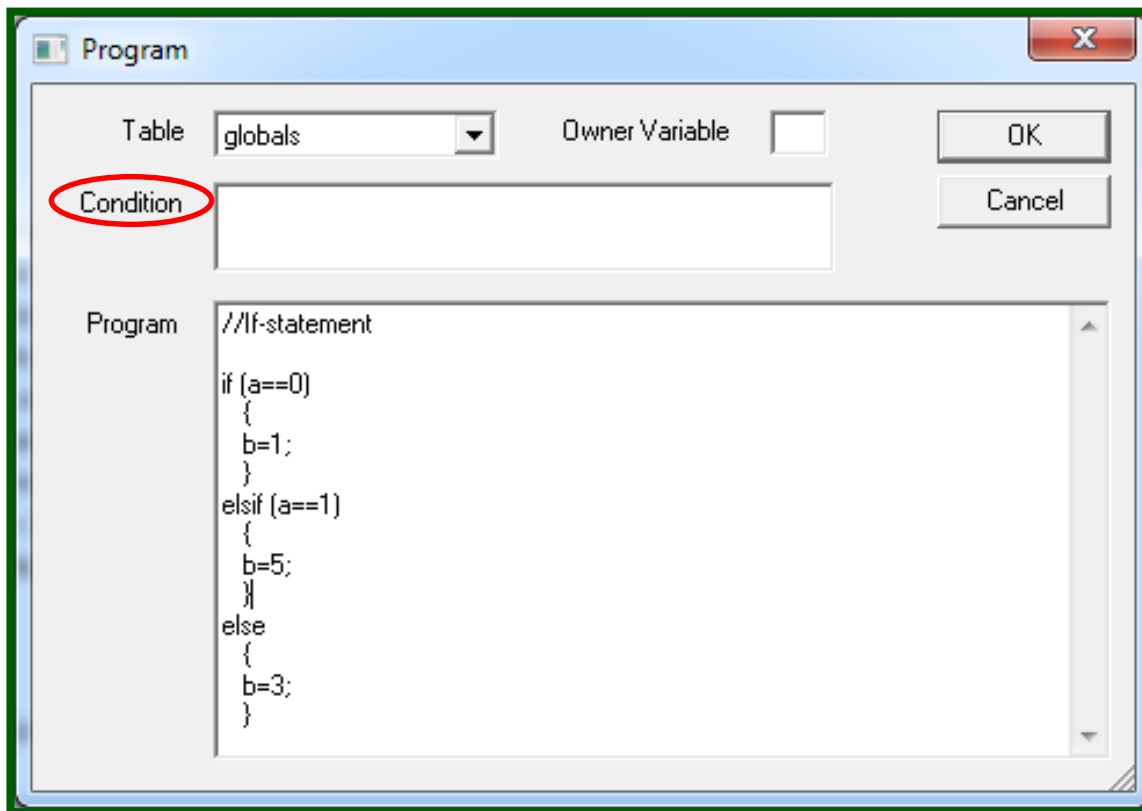


Examples of simple functions

- `result = if ( k < 5 | k >= 10, 1, 10);`
- `result = abs ( c + pi() );`
- `result = round ( a, 0.2);`
- `result = roundup ( random() * 5, 1 );`
- `result = exp ( randomgauss() );`
- `result = sqrt (b ^ 2);`
- `result = max ( ln ( x ), log ( y ) );`

Conditional execution in z-Tree

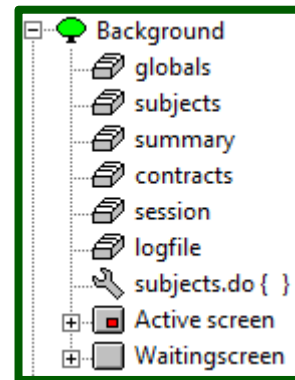
- If-statement for value assignment:
 - `result = if ( k < 5 | k >= 10, 1, 10);`
- If statement for code execution:



Structure of a treatment

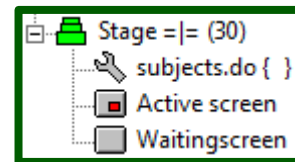
- Background

- Contains general settings
- Defines the tables used in the treatment
- Defines default Active Screen and Waiting Screen
- Contains programs which run at the beginning of a period



- Stages

- Correspond to the “screens” displayed to subjects
- Contain programs which run at the beginning of a stage



* Adapted from a z-Tree lecture by Maria Bigoni: http://www2.dse.unibo.it/bigoni/courses/ztree/Lecture_notes_handout.pdf, 31.08.2012, accessed on 19.12.2012.

Defining parameters in the Background

- Put general parameters into program in globals table in Background, e.g.:

Stage

Name: TradeScreenDay

Start:

- ☒ Wait for all
- ☐ Start if possible
- ☐ Start if...

Number of subjects in Stage:

- ☐ At most one per group in stage

Leave stage after timeout:

- ☐ If no input
- ☒ Yes
- ☐ No

Timeout: timeout

Program

Table: globals

Owner Variable:

Condition:

Program:

```
//Controls how long stages should be displayed
timeout=30;

//Sets display precision of prices
precision=1;

//Sets maximum price
maximumpricestock=10000;

//Sets limits for the number of transactions to be displayed
displaylimittransactions=2;
```

Always initialize every variable in the Background!

Defining parameters in the Background

- Put general parameters into program in globals table in Background, e.g.:

Item

Label

Variable: price

Layout: precision

☒ Input

Minimum: 0

Maximum: maximumpricestock

☐ Show value (value of variable or default)

☐ Empty allowed

Default

Program

Table: globals

Owner Variable

Condition

Program

```
//Controls how long stages should be displayed
timeout=30;

//Sets display precision of prices
precision=1;

//Sets maximum price
maximumpricestock=10000;

//Sets limits for the number of transactions to be displayed
displaylimittransactions=2;
```

Always initialize every variable in the Background!

Defining „switches“ in the Background

- Use variables to switch between test modes, and between different treatments within one .ztt file:

Program

Table: Owner Variable:

Condition:

Program: `//Switch to remove option market
optionmarketexists=0;

//Switch for test mode: 0=full, 1=skip profit display stage, 2=skip confirmation stage
testmode=0;`

Box

Name: ☒ With frame

Width [p/]: Distance to the margin [p/]:

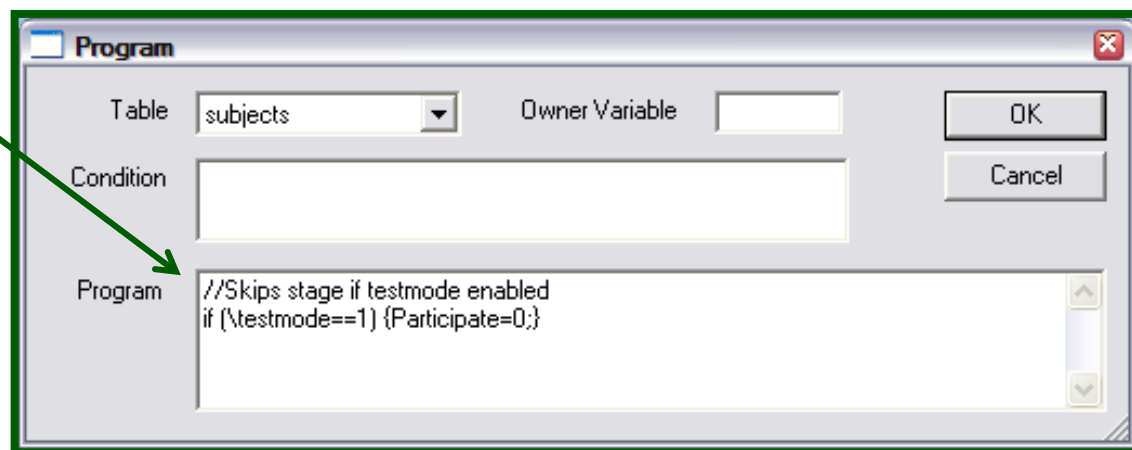
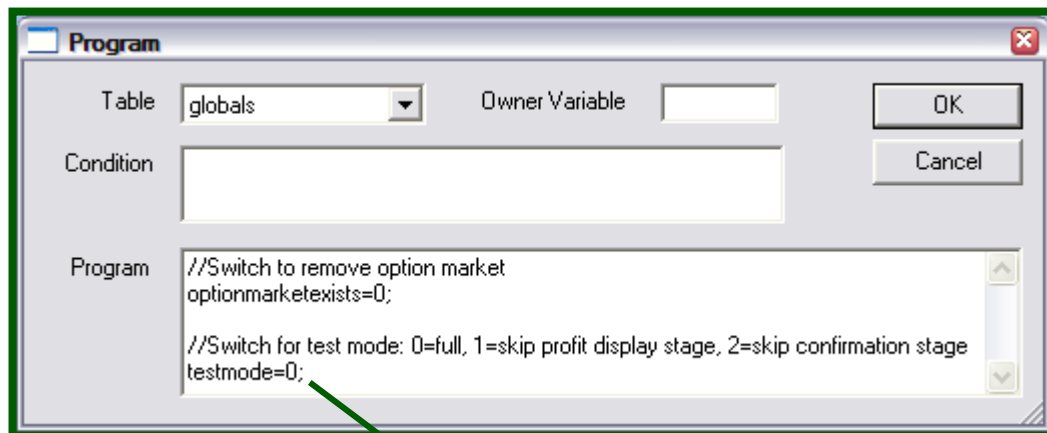
Height [p/]:

Adjustment of the remaining box: ☐ left ☐ top ☒ right ☐ bottom

Display condition:

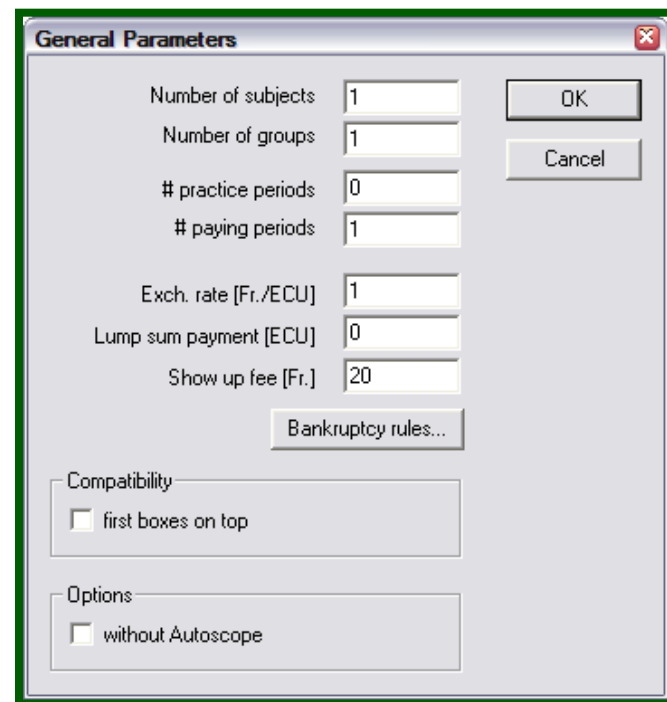
Defining „switches“ in the Background

- Use variables to switch between test modes, and between different treatments within one .ztt file:



Calculation of payouts: Background

- # practice periods
 - Periods without effect on profit
- # paying periods
 - Periods with profit
- Exchange rate
 - Value in real currency of 1 unit of the experimental currency (ECU)
- Lump sum payment
 - Value in ECU added to TotalProfit at the beginning of period 1 of the treatment
- Show up fee
 - Value in real currency added to subject payout at the beginning of period 1 of the session



The screenshot shows the 'General Parameters' dialog box with the following settings:

Parameter	Value
Number of subjects	1
Number of groups	1
# practice periods	0
# paying periods	1
Exch. rate [Fr./ECU]	1
Lump sum payment [ECU]	0
Show up fee [Fr.]	20

Buttons: OK, Cancel, Bankruptcy rules...

Compatibility:
☐ first boxes on top

Options:
☐ without Autoscope

Calculation of payouts: Bankruptcy

Bankruptcy Rules

Text "Do you invest your show up fee ?"

You have made a loss. Do you want to invest your show-up fee to compensate this loss?

"yes" Yes Show up fee is invested and experiment goes on

"no" No Message "BancruptShowupNo" appers in client's table"

Text "Do you want to go on"

You have made a loss. Do you want to continue?

"yes" Yes Message "BancruptMoreYes" appers in client's table"

"no" No Message "BancruptMoreNo" appers in client's table"

Text "Please wait until the experimenter unlocks your PC."

Please wait until the experimenter allows you to continue.

* The experimenter can either

- give permission to go on (give a credit)
- replace the subject (nullify payoffs)

Cancel

OK

Calculation of payouts: Profit/TotalProfit

- Table: subjects
 - Profit
 - Contains profit in a period, needs to be calculated
 - TotalProfit
 - Contains total profit up to period t , calculated automatically as the sum of the variables Profit of periods 1 to $(t - 1)$.
- Table: session
 - FinalProfit
 - Subject's profit excluding the show-up fee
 - MoneyAdded
 - Money added to a subject who faced bankruptcy but was allowed to continue
 - MoneyToPay
 - Equals FinalProfit plus ShowUpFee plus MoneyAdded

Link to
Calculation of payouts
in
Good Practice and Advice

Creating a stage

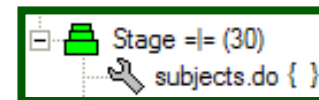
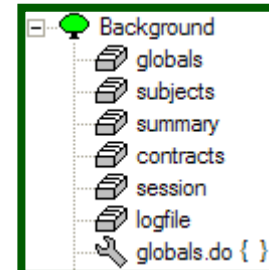
- Name is for documentation only; must be unique
- Start
- Timeout
- Active screen, Waitingscreen, Use background, Header
- Screen layout
 - Containers
 - Boxes
 - Absolute/relative size/position
 - Adjustment of the remaining box

Period structure

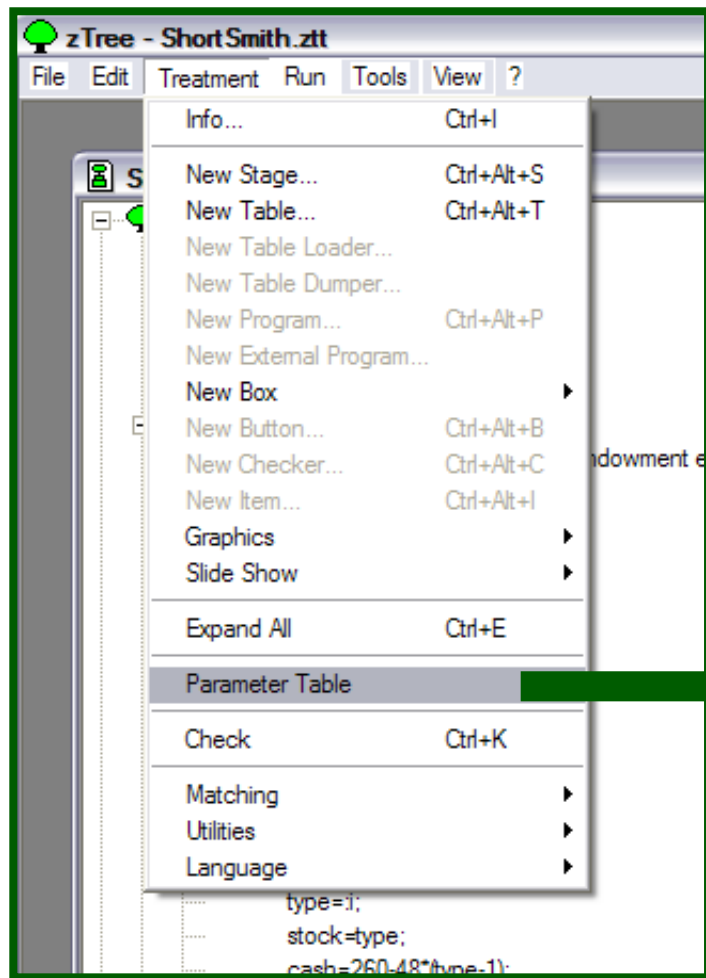
- Setting of standard variables
- Programs in the Background
- Specific parameter programs
- Role parameter programs
- Period parameter programs
- Programs at the beginning of a stage
- Programs in buttons (when clicked)
- Delayed sub-programs:
later () do { } / later () repeat { }

Running programs

- Program in the background
 - Runs after the setting of standard variables
- Program in a stage
 - Runs at beginning of stage, before checking Participate
- Program in a button
 - Program in subjects table runs only for the subject pressing the button
 - Runs after checkers
- Program started by later () do/repeat { } command
 - (Re-)Runs after pre-specified time



Parameter table



1. Specific parameters
(after programs in the
background)

2. Role parameters

	S1	S2	S3	S4
1	1	1	1	1
2	1	1	1	1

Green arrows point from the text labels to the table: one from '1. Specific parameters' to the '1' in row 1, column S1; one from '2. Role parameters' to the '1' in row 1, column S3; and one from '3. Period parameters & prompt' to the '2' in row 2, column S1.

3. Period parameters &
prompt

Loops in z-Tree

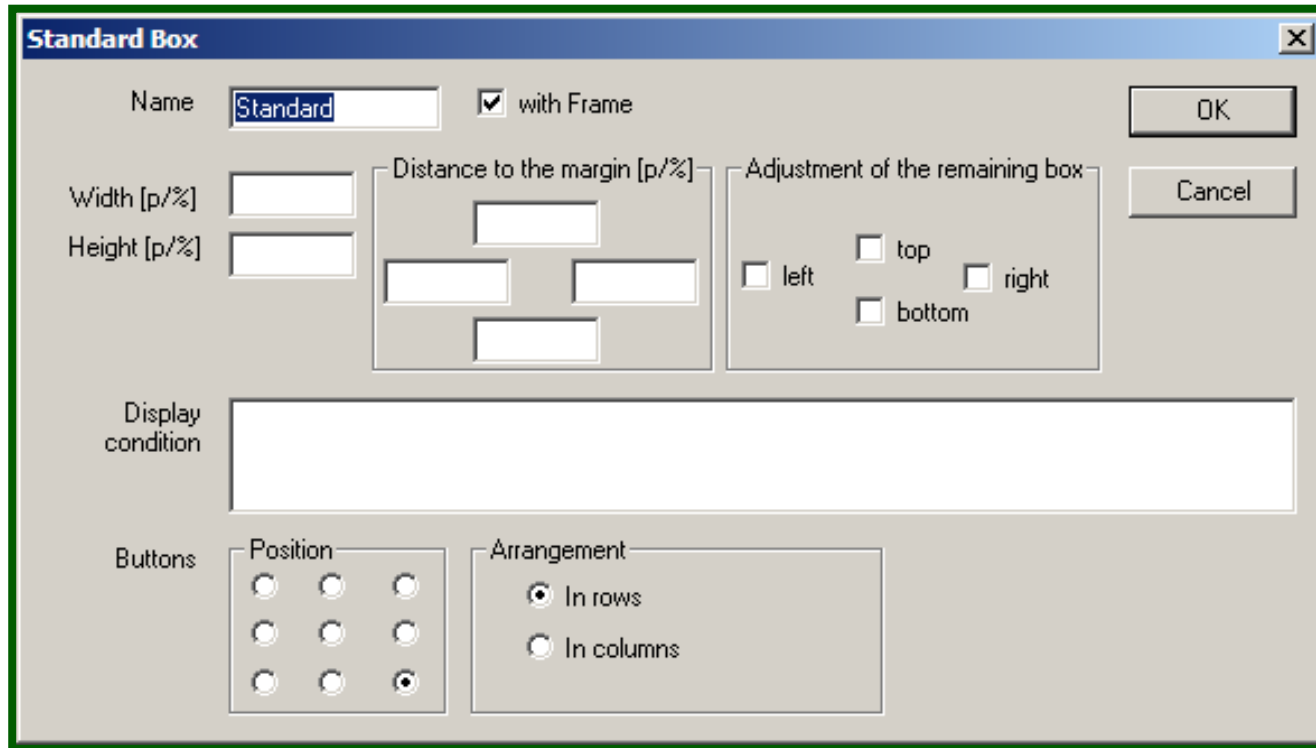
- **Loops through the records of a table:**
 - `contracts.do { player = :Subject; }`
- **Loops using `while ( )`:**
 - `while ( i <= 5 ) { i = i + 1; }`
- **Loops using `repeat { } while ( )`:**
 - `repeat { i = i + 1; } while ( i <= 5 );`
- **Loops using `iterator( ).do`:**
 - `iterator( i, 8, 12, 2 ).do { :player = i; }`
- **Loops in time using `later ( ) do/repeat { }`:**
 - `later ( 15 ) do` OR `repeat { }`
- While and repeat loops can be left with `<CTRL>-<ALT>-<F5>`

Screen Layout and Items

- Screen layout
- Information output
- Formatting output
- Information input
- Checking entries for correctness

Screen layout

- Screen layout can be controlled in box dialog:

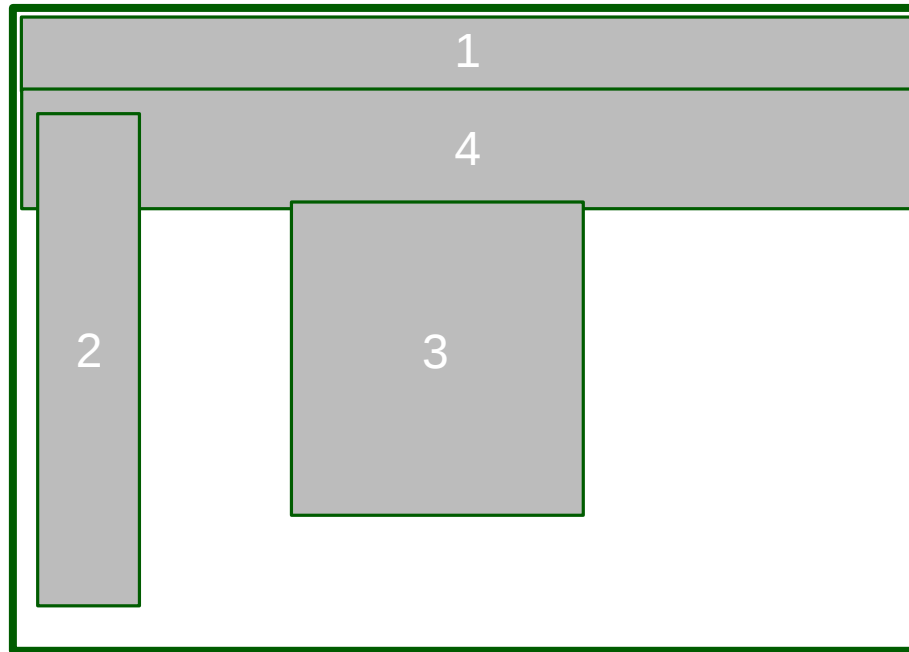


The image shows a 'Standard Box' dialog box with the following controls:

- Name:** A text field containing 'Standard'.
- with Frame:** A checked checkbox.
- Width [p/%]:** An empty text field.
- Height [p/%]:** An empty text field.
- Distance to the margin [p/%]:** A group box containing four empty text fields for top, bottom, left, and right margins.
- Adjustment of the remaining box:** A group box containing four checkboxes: 'left', 'top', 'right', and 'bottom'.
- Display condition:** A large empty text area.
- Buttons:** A group box containing two sub-sections:
 - Position:** A 3x3 grid of radio buttons, with the bottom-right button selected.
 - Arrangement:** Two radio buttons: 'In rows' (selected) and 'In columns'.
- OK** and **Cancel** buttons are located on the right side.

- See Demo_ScreenLayout.ztt for example layouts

Screen layout



Name	1	☒ with Frame
Width [p/%]		Distance to the margin [p/%]
Height [p/%]	10%	0p
		Adjustment of the remaining box
		☐ left ☒ top ☐ right
		☐ bottom

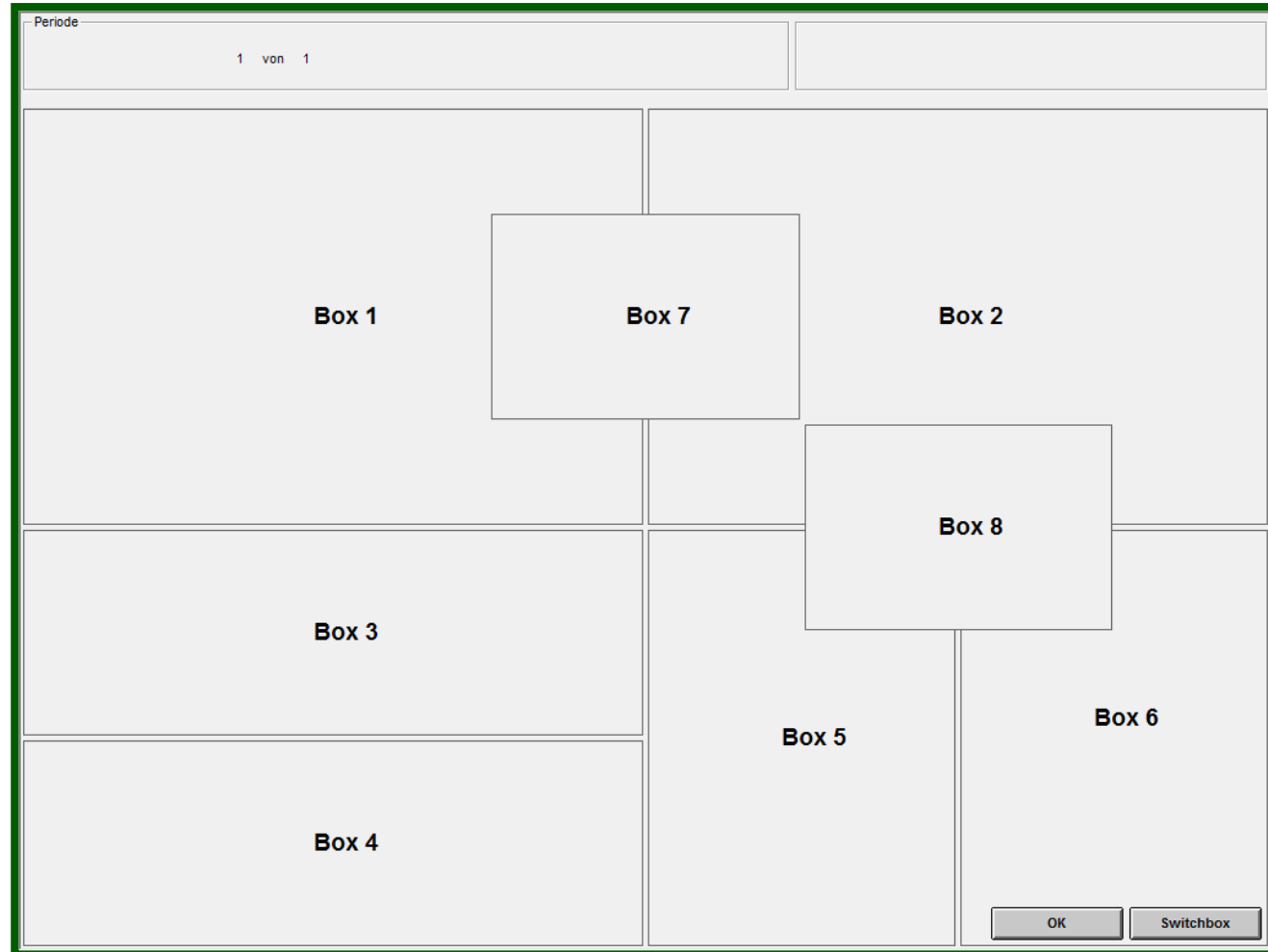
Name	2	☒ with Frame
Width [p/%]	50p	Distance to the margin [p/%]
Height [p/%]		20p
		20p
		20p
		Adjustment of the remaining box
		☐ left ☐ top ☐ right
		☐ bottom

Name	3	☒ with Frame
Width [p/%]	30%	Distance to the margin [p/%]
Height [p/%]	50%	
		Adjustment of the remaining box
		☐ left ☐ top ☐ right
		☒ bottom

Name	4	☒ with Frame
Width [p/%]		Distance to the margin [p/%]
Height [p/%]		
		Adjustment of the remaining box
		☐ left ☐ top ☐ right
		☐ bottom

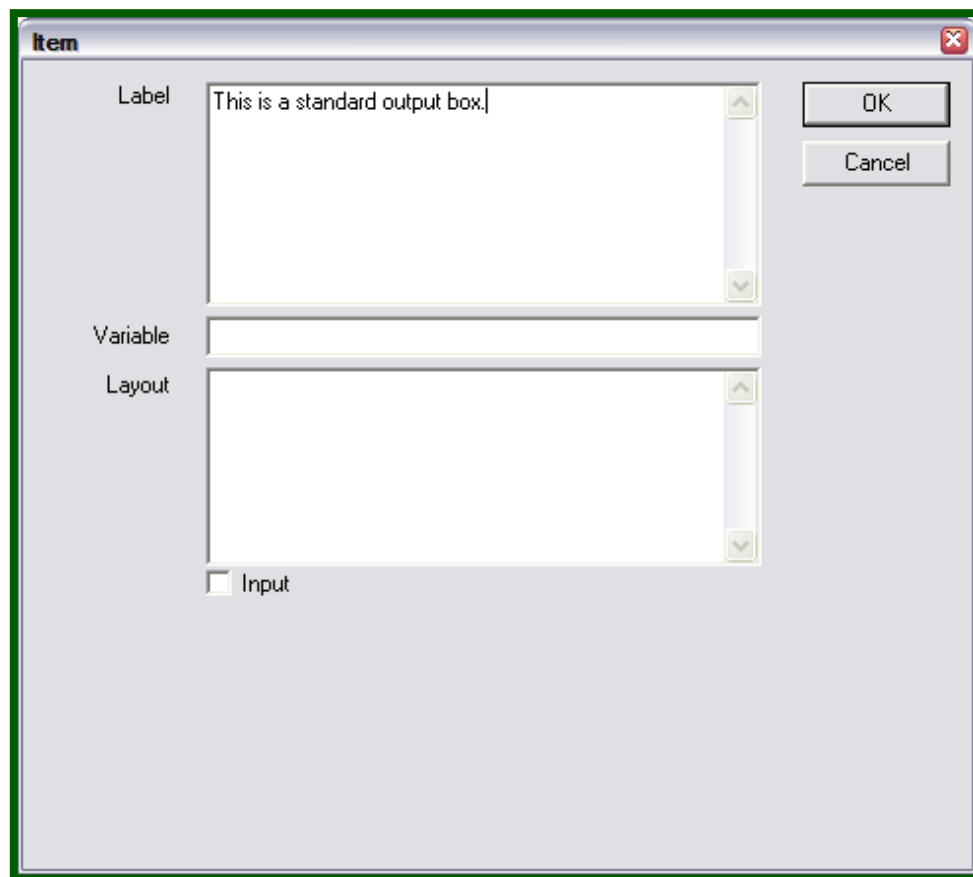
Exercise - Screen layout*

- Program a treatment which displays (approximately) the following:



* Inspired by lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

Items: Text output



Item

Label: This is a standard output box.

Variable:

Layout:

☐ Input

OK

Cancel

- Result: This is a standard output box.

Items: Variable output

Item

Label: This item displays the subject number:

Variable: Subject

Layout: 1

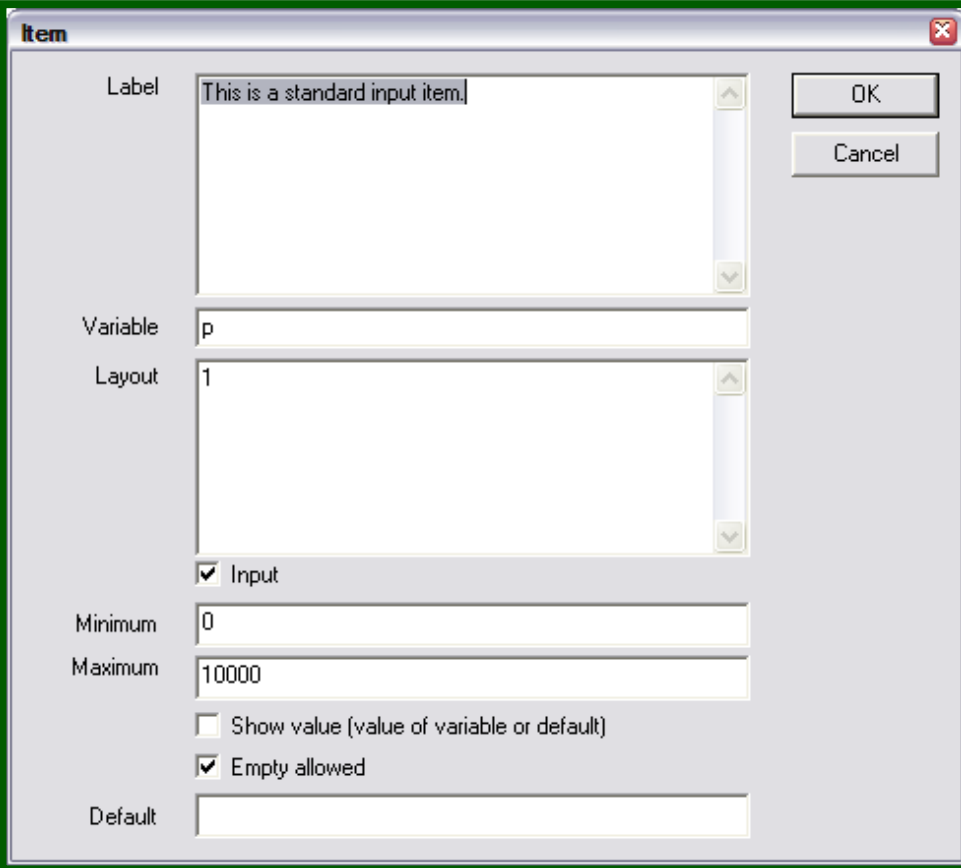
☐ Input

OK

Cancel

- Result: This item displays the subject number: 1

Items: Variable input



The screenshot shows a dialog box titled "Item" with a close button (X) in the top right corner. The dialog contains several fields and checkboxes:

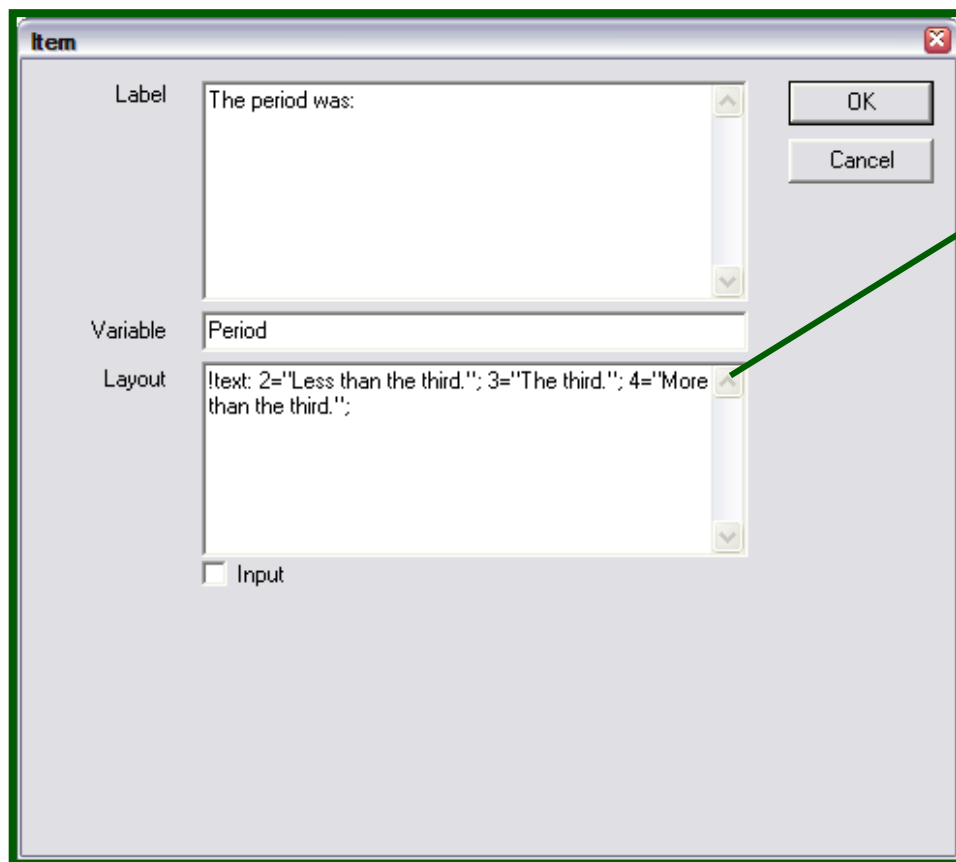
- Label:** A text field containing "This is a standard input item." with a vertical scrollbar on the right.
- Variable:** A text field containing "p".
- Layout:** A text field containing "1" with a vertical scrollbar on the right.
- Input:** A checked checkbox.
- Minimum:** A text field containing "0".
- Maximum:** A text field containing "10000".
- Show value (value of variable or default):** An unchecked checkbox.
- Empty allowed:** A checked checkbox.
- Default:** An empty text field.
- Buttons:** "OK" and "Cancel" buttons are located to the right of the "Label" field.

- **Result:**



The screenshot shows the resulting input field in the z-Tree interface. It consists of a text label "This is a standard input item." followed by a light blue rectangular input box. The entire element is enclosed in a green border.

Conditional output



Item

Label: The period was:

Variable: Period

Layout: !text: 2="Less than the third."; 3="The third."; 4="More than the third.;"

☐ Input

OK

Cancel

Layout:

```
!text: 2="Less than the  
third."; 3="The third.";  
4="More than the third.";
```

Attention:

The first text version that is being displayed sets the size of the item. If the variable value (and accordingly the text) changes, longer text may not display correctly. Therefore, add enough blank space after each option to display even the longest.

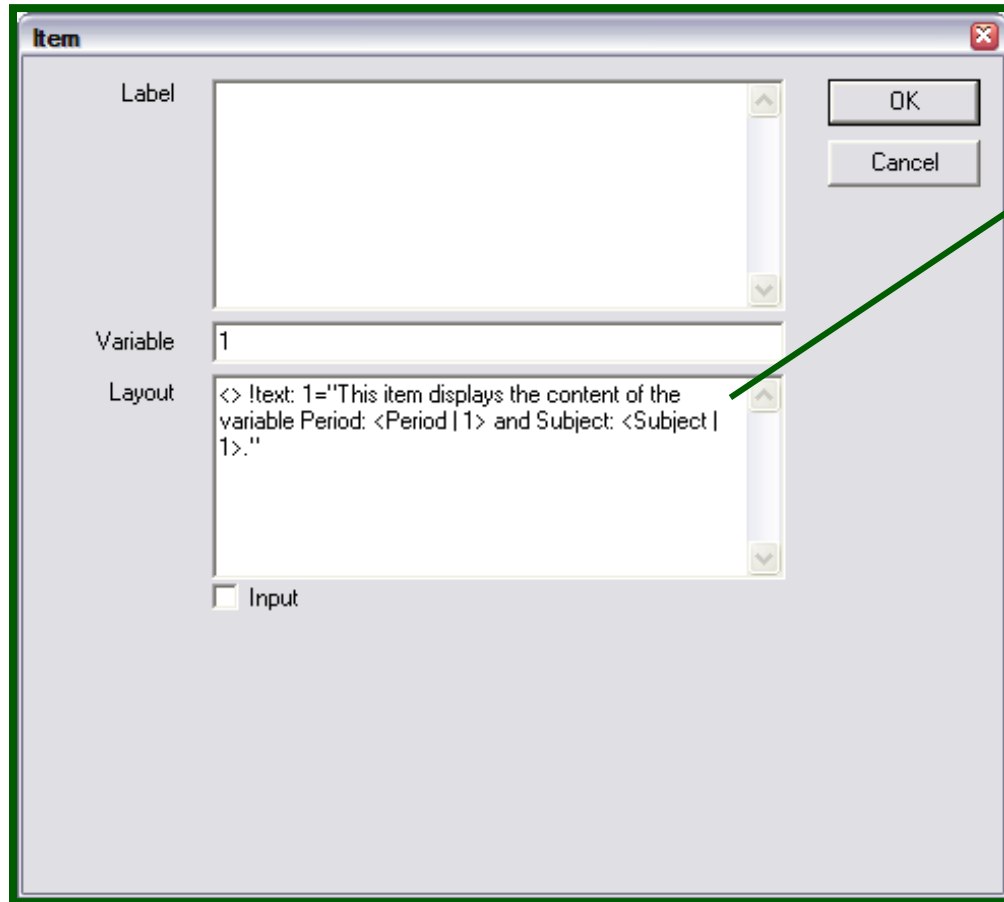
• Results:

Period 1+2: The period was: Less than the third.

Period 3: The period was: The third.

Period 4+: The period was: More than the third.

Inserting variables into text



The screenshot shows the 'Item' dialog box with the following fields:

- Label:** A large empty text area.
- Variable:** A text field containing the value '1'.
- Layout:** A text area containing the code: `<> !text: 1="This item displays the content of the variable Period: <Period | 1> and Subject: <Subject | 1>."`
- Input:** A checkbox that is currently unchecked.
- Buttons:** 'OK' and 'Cancel' buttons are located in the top right corner.

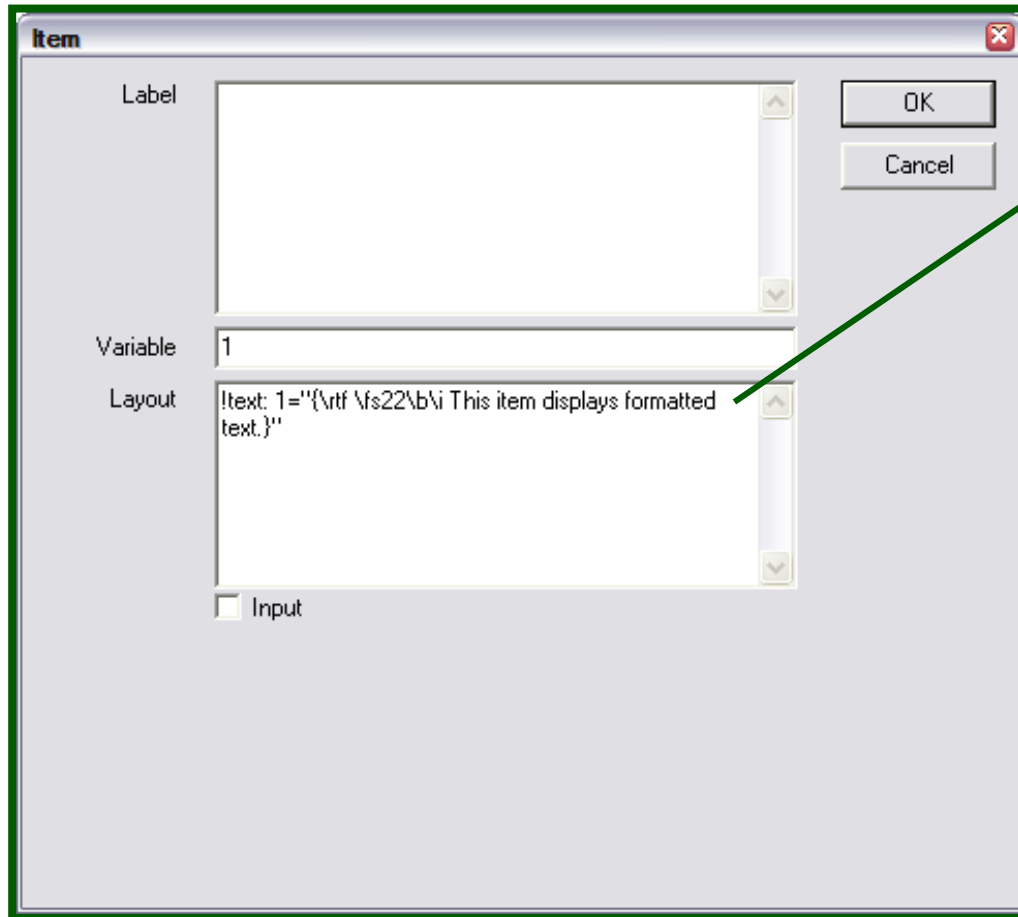
A green arrow points from the 'Layout' field to the 'OK' button.

Layout:

```
<> !text: 1="This item
displays the content of
the variable Period:
<Period | 1> and Subject:
<Subject | 1>."
```

- **Result:** This item displays the content of the variable Period: 1 and Subject: 2.

Formatted text



Item

Label

Variable

Layout

☐ Input

OK

Cancel

Layout:

```
!text: 1="{\\rtf \\fs22\\b\\i  
This item displays  
formatted text.}"
```

- Result: ***This item displays formatted text.***

Variables embedded in colored text

Item

Label

Variable: \NumSubjects

Layout: <> !text: 1="{\rtf {\colortbl; \red255\green0\blue0; \red0\green0\blue255;\red0\green0\blue0;} \fs24 This is a \cf2 colorfully formatted \cf3 output box displaying the variables votes: \cf1 <\votes | 1>\cf3 ; and NumSubjects: \cf1 <\NumSubjects | 1>\cf3 .}"

☐ Input

OK

Cancel

Layout:

```
<> !text: 1="{\rtf
{\colortbl;
\red255\green0\blue0;
\red0\green0\blue255;\red
0\green0\blue0;} \fs24
This is a \cf2 colorfully
formatted \cf3 output box
displaying the variables
votes: \cf1 <\votes |
1>\cf3 ; and NumSubjects:
\cf1 <\NumSubjects |
1>\cf3 .}"
```

- Result:

This is a colorfully formatted output box displaying the variables votes: 0; and NumSubjects: 1.

Conditional formatting

Item

Label

Variable: \NumSubjects

Layout: <> !text: 1="{\rtf \fs24 <\Period | !text: 1="\i " 2="\b ">Test}"

☐ Input

OK

Cancel

Layout:

```
<> !text: 1="{\rtf \fs24  
<\Period | !text: 1="\i "  
2="\b ">Test}"
```

• Result:

Period 1:

Test

Period 2+:

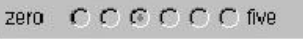
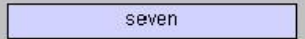
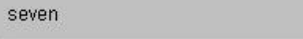
Test

Rich text formatting

Switch	Effect
<code>\tab</code>	tabulator
<code>\par</code>	new paragraph
<code>\line</code>	new line
<code>\bullet</code>	thick dot·
<code>\ql</code>	aligned to left
<code>\qr</code>	aligned to right
<code>\qc</code>	centered
<code>\b</code>	bold
<code>\b0</code>	not bold
<code>\i</code>	italic
<code>\i0</code>	not italic
<code>\sub</code>	small and inferior numbers (index)
<code>\super</code>	small and superior numbers (exponent)
<code>\strike</code>	crossed through
<code>\ul</code>	underline
<code>\ul0</code>	do not underline
<code>\cfn</code>	Text color. n is the index of the color table which is defined by colortbl. See example.
<code>\fsn</code>	Font size in units of half a dot. The font size must be explicitly given, otherwise it is larger (24) than usual in z-Leaf.

Alternative formats

Examples of item layouts

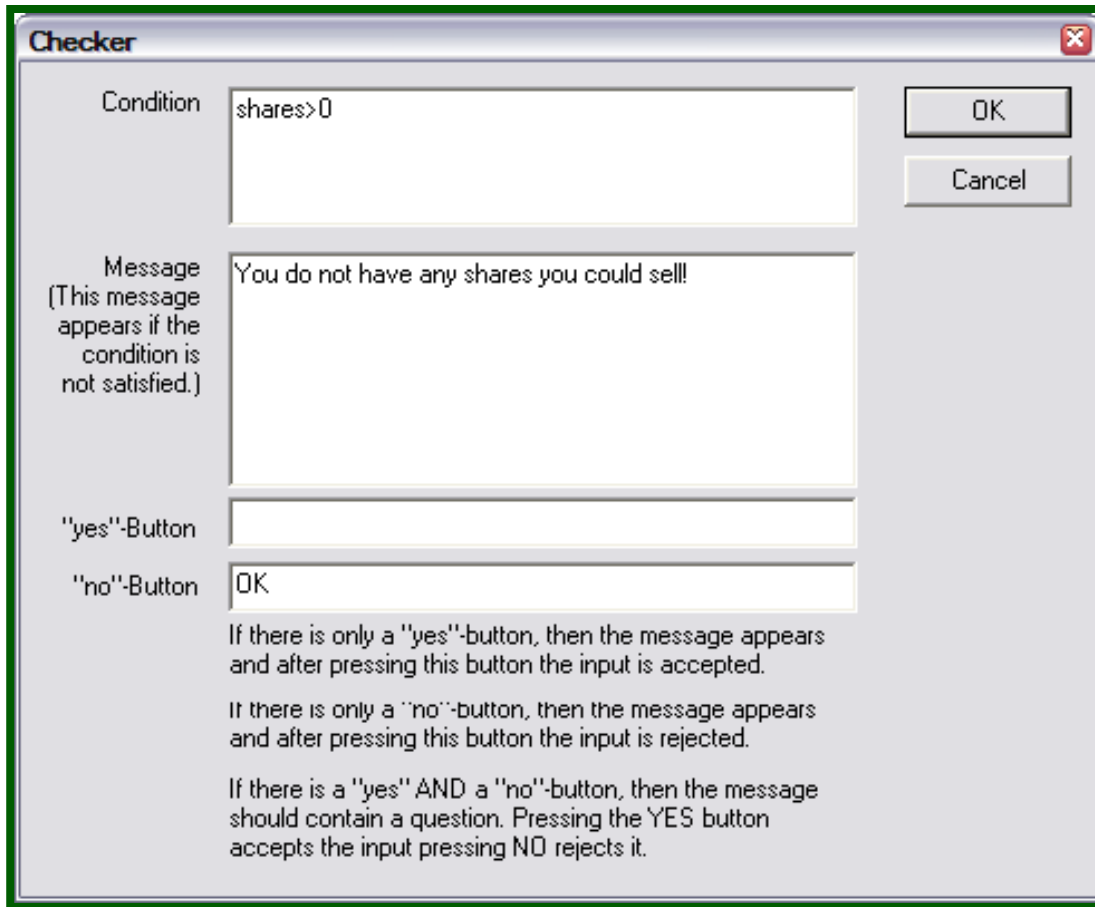
Layout	input variable	output variable
2		
!radio: 1 = "86.8"; 24 = "102.8";		
!radioline: 0="zero";5="five"; 6;		
!slider: 0 = "A"; 100= "B"; 101;		
!scrollbar: 0="L ";100= "R ";101;		
!checkbox:1="check me";		
!text: 1= "one"; 2 = "two"; 3 = "three"; 4 = "four"; 5 = "five"; 6 = "six"; 7 = "seven"; 8 = "eight"; 9 = "nine"; 10 = "ten";		
!button: 1 = "accept"; 0 = "reject";		

General notes on formatting

- Theoretically, any RTF code (e.g. saved in Word) should display correctly
- Variable refresh
 - Put text in layout (as opposed to label) section to allow variables to refresh
 - Screen space is reserved for initial value of variables
- Change font size generally through z-leaf command line switch `/fontsize`

Checking entries for correctness

- Select a button
- Click Treatment-New Checker...



The screenshot shows a dialog box titled "Checker" with a standard Windows-style title bar (minimize, maximize, close buttons). The dialog is divided into several sections:

- Condition:** A text input field containing the expression "shares>0".
- Message:** A text input field containing the text "You do not have any shares you could sell!". To the left of this field is a label "Message" followed by a smaller text "(This message appears if the condition is not satisfied.)".
- "yes"-Button:** An empty text input field.
- "no"-Button:** A text input field containing the text "OK".

On the right side of the dialog, there are two buttons: "OK" and "Cancel".

At the bottom of the dialog, there is explanatory text:

If there is only a "yes"-button, then the message appears and after pressing this button the input is accepted.

If there is only a "no"-button, then the message appears and after pressing this button the input is rejected.

If there is a "yes" AND a "no"-button, then the message should contain a question. Pressing the YES button accepts the input pressing NO rejects it.

Tables and Scope

- Built-in tables
- Table functions
- Scope environment
- User-generated tables

General concepts

- Lifetime
- Program execution
- Used variables
- Columns
- Columns 1 and 2
- Binary variable format

Table

Name: session

Lifetime:

- ☐ Period
- ☐ Treatment
- ☒ Session

Program execution:

- ☐ When first subject enters stage
- ☒ Always when a subject enters stage
- ☐ When last subject enters stage

Used Variables (comma separated):

OK Cancel

1subjects	Period	Subject	Group	Profit	TotalProfit	Participate	Money	Stock	
1subjects		1	1	1	0	0	0	6392	10
1subjects		1	2	1	0	0	0	320	7
1subjects		1	3	1	0	0	0	6408	3
1subjects		1	4	1	0	0	0	64	0
2subjects	Period	Subject	Group	Profit	TotalProfit	Participate	Money	Stock	
2subjects		1	1	1	18.5066667	18.5066667	0	1388	5
2subjects		1	2	1	1.2666667	1.2666667	0	95	5
2subjects		1	3	1	25.96	25.96	0	1947	0
2subjects		1	4	1	2.3066667	2.3066667	0	173	0
2subjects	Period	Subject	Group	Profit	TotalProfit	Participate	Money	Stock	
2subjects		2	1	1	6.69333333	25.2	0	1890	4
2subjects		2	2	1	3.21333333	4.48	0	336	

subjects table

Period	Subject	Group	Profit	TotalProfit	Participate
1	1	1	0	0	1
1	2	1	0	0	1
1	3	1	0	0	1
1	4	1	0	0	1

Table subjects

- Life: Period
- One row per subject
- Main table for subject data
- Default table for items (input and output)
- Pre-defined variables: Period, Subject, Group, Profit, TotalProfit, Participate, Priority, LeaveStage

1 subjects	Period	Subject	Group	Profit	TotalProfit	Participate	Money	Stock	
1 subjects		1	1	1	18.5066667	18.5066667	0	1388	5
1 subjects		1	2	1	1.26666667	1.26666667	0	95	5
1 subjects		1	3	1	25.96	25.96	0	1947	0
1 subjects		1	4	1	2.30666667	2.30666667	0	173	0
1 subjects	Period	Subject	Group	Profit	TotalProfit	Participate	Money	Stock	
1 subjects		2	1	1	6.69333333	25.2	0	1890	4
1 subjects		2	2	1	3.21333333	4.48	0	336	4
1 subjects		2	3	1	0	25.96	0	1947	0
1 subjects		2	4	1	0	2.30666667	0	173	0

Table globals

- Life: Period
- One row only
- Main table for general data for the treatment
- Can be reached by scope operator \
- Pre-defined variables: Period, NumPeriods, RepeatTreatment

1	globals	Period	NumPeriods	RepeatTreatment	StartTime	TimeAuction	TimeProfit
1	globals	1	1	0	14483,803	600	0
2	globals	Period	NumPeriods	RepeatTreatment	StartTime	TimeAuction	TimeProfit
2	globals	1	15	0	15368,391	240	30
2	globals	Period	NumPeriods	RepeatTreatment	StartTime	TimeAuction	TimeProfit
2	globals	2	15	0	17116,102	240	30

Table session

- Life: Session
- One row per subject
- Main table for payment information
- Can be used to transfer data to questionnaire and between treatments
- Pre-defined variables: Subject, FinalProfit, ShowUpFee, ShowUpFeeInvested, MoneyAdded, MoneyToPay, MoneyEarned

2	session	Subject	FinalProfit	ShowUpFee	ShowUpFeeInvested	MoneyAdded	MoneyToPay
2	session	1	44,0933333	0	0	0	44,0933333
2	session	2	35,2	0	0	0	35,2
2	session	3	2,05333333	0	0	0	2,05333333
2	session	4	21,08	0	0	0	21,08

Table summary

- Life: Treatment
- One row per Period
- Can be used to save summary data for experimenter view or later analysis
- Pre-defined variables: Period

1	summary	Period	value	avP	nTrades	dividend	LastP
1	summary	1	24	43,1176471	221	60	40
2	summary	Period	value	avP	nTrades	dividend	LastP
2	summary	1	360	149,8	75	60	100
2	summary	2	336	191,3	30	60	260
2	summary	3	312	300,222222	18	28	335
2	summary	4	288	288,529412	17	28	280

Table contracts

- Life: Period
- No pre-defined number of rows
- Table for transaction data
- Default table for contract boxes
- Pre-defined variables: Period

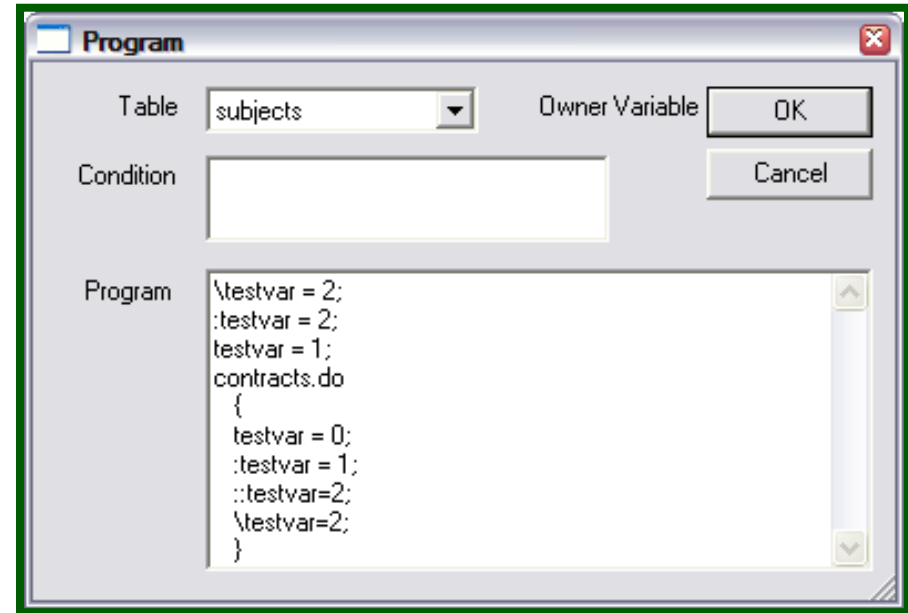
1	contracts	Period	Seller	Buyer	Group	p	Traded	contractID	tradeID	Maker
1	contracts	1	10	8	1	20	1	1	1	10
1	contracts	1	5	8	1	40	1	2	11	5
1	contracts	1	4	-2	1	120	0	3	0	4
1	contracts	1	-2	8	1	100	0	4	0	8
1	contracts	1	6	8	1	30	1	5	2	6
1	contracts	1	4	6	1	40	1	6	6	6
1	contracts	1	2	8	1	40	1	7	3	2
1	contracts	1	9	3	1	40	1	8	4	9
1	contracts	1	3	6	1	60	1	11	9	6
1	contracts	1	-2	2	1	30	0	12	0	2
1	contracts	1	1	2	1	130	1	13	30	1

Table functions

- Can be preceded by table name, e.g. `subjects.sum()`
- Examples:
 - `sum ( [condition] , variable );`
 - `average ( [condition] , variable );`
 - `product ( [condition] , variable );`
 - `minimum / median / maximum ( [condition] , variable );`
 - `find ( [condition] , variable );`
 - `count ( [condition] );`
 - `regressionslope ( [condition] , x , y );`
 - `stddev ( [condition] , variable );`

Programs, tables and scope operators

- Programs run in a table
- Programs can contain commands running in different table
- Structure here is:
 - globals
 - subjects
 - contracts



Scope operators:
: one step up
\ all steps up to
„globals“ table

Programs, tables and scope operators

- Programs run for all records in a table (unless restricted by conditions)

The screenshot shows the 'Program' dialog box in the z-Tree software. The 'Table' dropdown is set to 'globals'. The 'Condition' field is empty. The 'Program' field contains the following code:

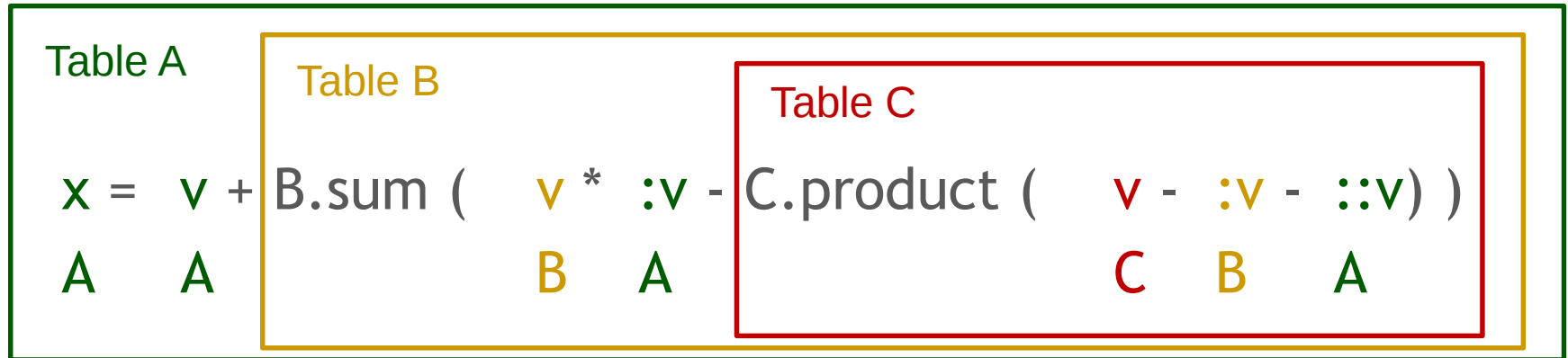
```
//Randomly assigns endowment type
endowmenttype=1;
//Repeats as often as there are endowment types
while (endowmenttype<=endowmenttypes)
{
  i=1;
  //Repeats as often as one endowment type is assigned to a trader (e.g. 10 traders, 2 types = 20 times)
  while (i<=endowmentspertype)
  {
    j=roundup(random()*(NumSubjects-(endowmenttype-1)*endowmentspertype-(i-1)),1);
    k=1;
    l=0;
    done=0;
    while (k<=NumSubjects&done==0)
    {
      if (subjects.find(Subject==k,InitialMoney)==0) {l=l+1;}
      if (j==l)
      {
        subjects.do
        {
          if (Subject==k)
          {
            InitialMoney=endowments.find(type==endowmenttype,cash);
            InitialStock=endowments.find(type==endowmenttype,stock);
            \done=1;
          }
        }
        k=k+1;
      }
      i=i+1;
    }
    endowmenttype=endowmenttype+1;
  }
}
```

Annotations with arrows point to specific parts of the code and the dialog box:

- A green arrow points from the 'Table' dropdown (set to 'globals') to a box labeled 'Variables in table „globals“'.
- A green arrow points from the 'Condition' field to the same box.
- A green arrow points from the 'Program' field to the same box.
- A green arrow points from the 'Program' field to a box labeled 'Variables in table „endowments“'.
- A red arrow points from the 'Program' field to the same box.
- A green arrow points from the 'Program' field to a box labeled 'Variables in table „subjects“'.
- A yellow arrow points from the 'Program' field to the same box.
- A yellow box labeled 'Command run in table „subjects“' has two yellow arrows pointing to the 'subjects.find' and 'subjects.do' lines in the code.

Programs, tables and scope operators

- Example: (Program runs in table A)



- Function same() as a special case
- If variable only defined in one table, scope operator may be omitted (dangerous!)
- Last period's tables: OLDsubjects, etc.

The complete scope reference

table	Program lies in	Owner var.	Cond.	Records for which the program is run	Records accessible through scope operator
globals			without	Record of globals table	-
			with	If cond. is met, record of globals table	-
subjects			without	All records of subjects table	globals
			with	Records of subjects table for which cond. is met	globals
summary			without	Record of current period	globals
			with	Record of current period, if cond. is met	globals
contracts	stage	without	without	All records of contracts table	globals
		without	with	All records of contracts table that meet cond.	globals
		with	without	All records of contracts table. The records are worked through subject for subject. The records of a subject are those records for which the Owner variable equals the Subjects variable.	Record of owner in subjects table, globals
		with	with	Same as above, except that the record of cond. also needs to be met	Record of owner in subjects table, globals
contracts	standard box, grid box, contract grid box		without	All records of contracts table	Record in subjects table of subject who has clicked the button globals
			with	All records of contracts table that meet cond.	ditto
contracts	contract creation box, contract list box		without	All new records of contracts table in the case of the contract creation box, and the selected record in the case of the contract list box. (To run the program for all records cond. has to be set to TRUE)	ditto
			with	All records of contracts table that meet cond.	ditto

Source: Reference manual, p. 59.

User generated tables

- Can serve different purposes
 - Make it easier to find information
 - Prevent duplication of information
- Allow for use of table functions
- Targeted import and export of data

Link to
Dividends
in
Exercises

Make maximum use of tables

- Use instead of arrays for better output format
- Use table loader to quickly import new parameters
- Use to transfer data between treatments (instead of OLDsubjects, etc.)
- Use table functions to quickly calculate summary information
- Table lifetime
 - Period (contracts, globals, subjects, ...)
 - Treatment (summary, ...)
 - Session (session, ...)

Matching Schemes

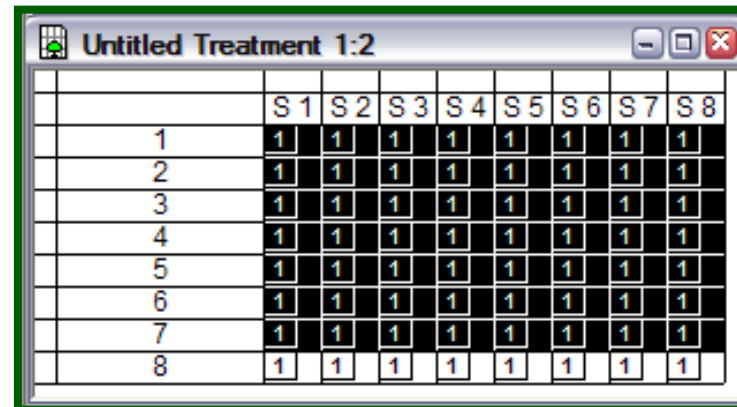
- Built-in matching capabilities
- User-generated matching

Matching schemes

- Matching in z-Tree via group membership
- Group variable pre-defined in subjects table
- Groups are defined in Specific parameters in Parameter table
- Groups can be modified using Group variable in subjects table
- Remember period structure

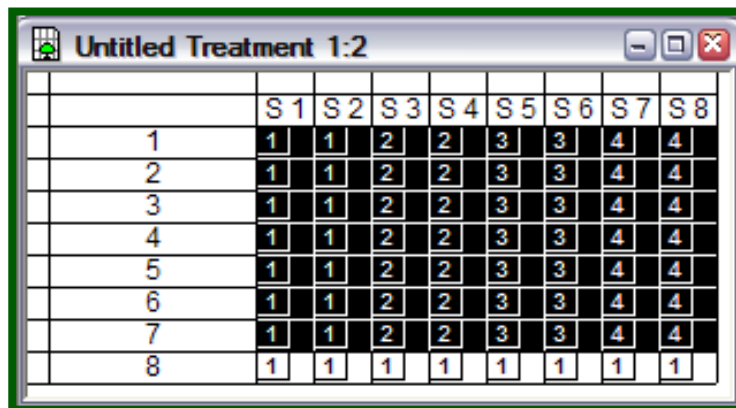
Group matching

- Select desired periods in parameter table:

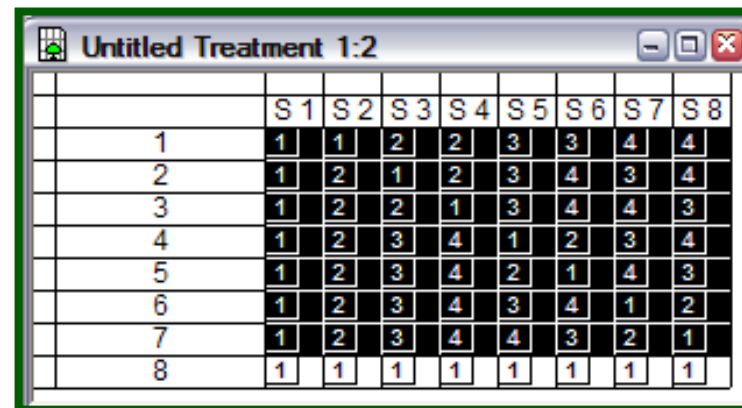


	S 1	S 2	S 3	S 4	S 5	S 6	S 7	S 8
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1

- Choose matching procedure from Treatment menu:
 - Partner:
 - Absolute stranger:



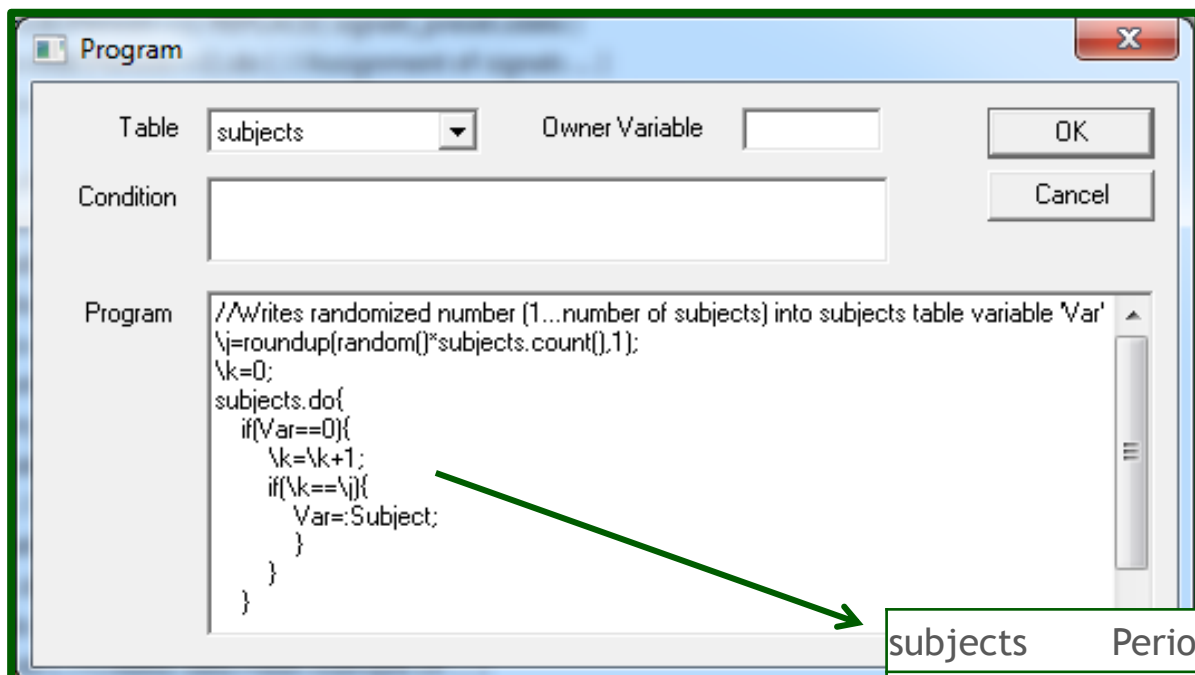
	S 1	S 2	S 3	S 4	S 5	S 6	S 7	S 8
1	1	1	2	2	3	3	4	4
2	1	1	2	2	3	3	4	4
3	1	1	2	2	3	3	4	4
4	1	1	2	2	3	3	4	4
5	1	1	2	2	3	3	4	4
6	1	1	2	2	3	3	4	4
7	1	1	2	2	3	3	4	4
8	1	1	1	1	1	1	1	1



	S 1	S 2	S 3	S 4	S 5	S 6	S 7	S 8
1	1	1	2	2	3	3	4	4
2	1	2	1	2	3	4	3	4
3	1	2	2	1	3	4	4	3
4	1	2	3	4	1	2	3	4
5	1	2	3	4	2	1	4	3
6	1	2	3	4	3	4	1	2
7	1	2	3	4	4	3	2	1
8	1	1	1	1	1	1	1	1

Randomly ordering table entries

- Useful for random assignment of groups or endowment types



The screenshot shows the 'Program' dialog box in z-Tree. The 'Table' dropdown is set to 'subjects'. The 'Program' text area contains the following script:

```
//Writes randomized number (1...number of subjects) into subjects table variable 'Var'
\j=roundup(random()*subjects.count(),1);
\k=0;
subjects.do{
  if(\Var==0){
    \k=\k+1;
    if(\k==\j){
      \Var=:Subject;
    }
  }
}
```

A green arrow points from the script to a table showing the results of the random assignment:

subjects	Period	Subject	Var
subjects	1	1	3
subjects	1	2	1
subjects	1	3	2

Modulo function

- Usage: $\text{mod}(x,y)$
- Returns the remainder of division of x by y
- Example: Group assignment
 - 3 groups
 - Based on variable SortVar

- Command:

Group=
 $\text{mod}(\text{SortVar}-1,3)+1$

SortVar	$\text{mod}(\text{SortVar}-1,3)$	Group
1	0	1
2	1	2
3	2	3
4	0	1
5	1	2
6	2	3

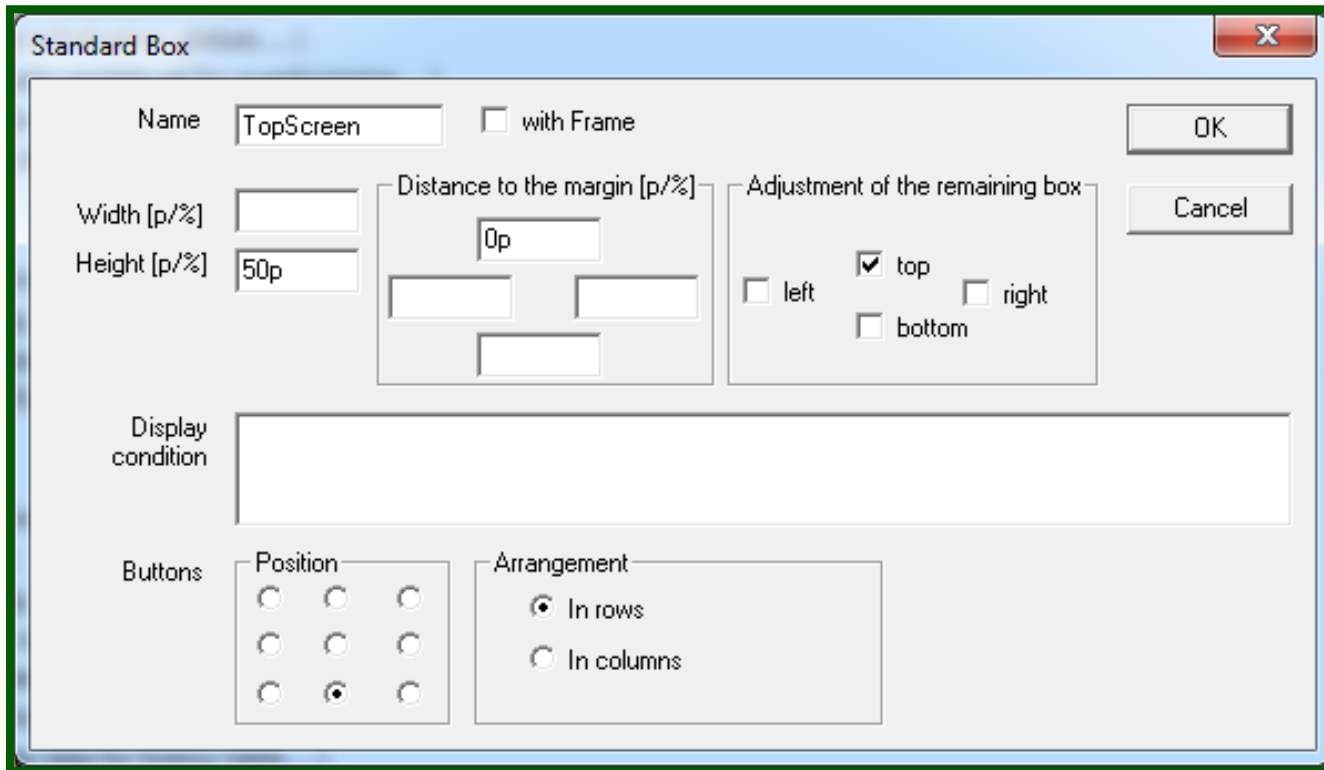
Box Types

- Container box
- Contract list box
- Contract creation box
- Grid box
- Calculator button box
- Chat box

Box types

Container box: 

- Used to organize other boxes on the screen



The screenshot shows the 'Standard Box' dialog box with the following settings:

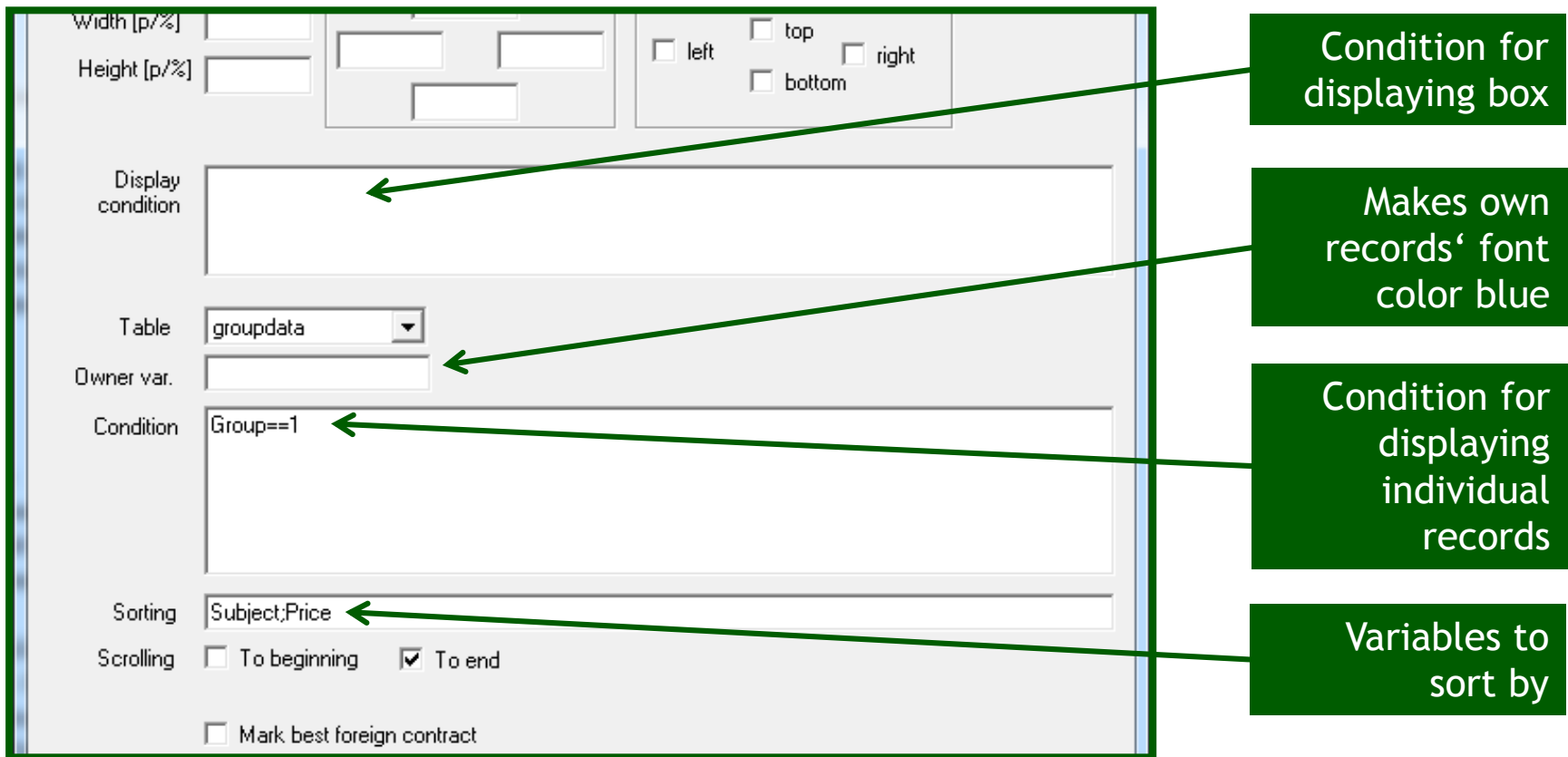
- Name:** TopScreen
- with Frame:** ☐
- Width [p/%]:** [empty]
- Height [p/%]:** 50p
- Distance to the margin [p/%]:** 0p
- Adjustment of the remaining box:**
 - ☐ left
 - ☒ top
 - ☐ right
 - ☐ bottom
- Display condition:** [empty text box]
- Buttons:**
 - Position:** A 3x3 grid of radio buttons with the center button selected.
 - Arrangement:**
 - ☒ In rows
 - ☐ In columns

OK and Cancel buttons are located on the right side of the dialog.

Box types

Contract list box:

- Used to display records from a z-Tree table:

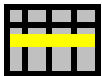


The screenshot shows the configuration dialog for a Contract list box. It includes fields for Width [p/%], Height [p/%], and position checkboxes (left, top, right, bottom). The 'Display condition' field is empty. The 'Table' dropdown is set to 'groupdata'. The 'Owner var.' field is empty. The 'Condition' field contains 'Group==1'. The 'Sorting' field contains 'Subject;Price'. The 'Scrolling' section has 'To beginning' unchecked and 'To end' checked. There is a checkbox for 'Mark best foreign contract'.

Annotations with arrows pointing to specific fields:

- Condition for displaying box (points to Display condition)
- Makes own records' font color blue (points to Table)
- Condition for displaying individual records (points to Condition)
- Variables to sort by (points to Sorting)

Box types

Contract list box: 

- Code:

```
History: groupdata( Group==1 )
  ☐ Periode: OUT( Period )
  ☐ Spielerzahl: OUT( Subjects )
  ☐ Durchschnittl. Beitrag/Kopf: OUT( Contribution )
  ☐ Durchschnittl. Ertrag/Kopf: OUT( Payoff )
  ☐ Durchschnittl. Endpunkte- stand: OUT( Wealth )
```



- Output:

Periode	Spielerzahl	Durchschn. Beitrag/Kopf	Durchschn. Ertrag/Kopf	Durchschn. Endpunkte- stand
1	1	12.00	7.20	15.20
2	1	5.00	3.00	18.00

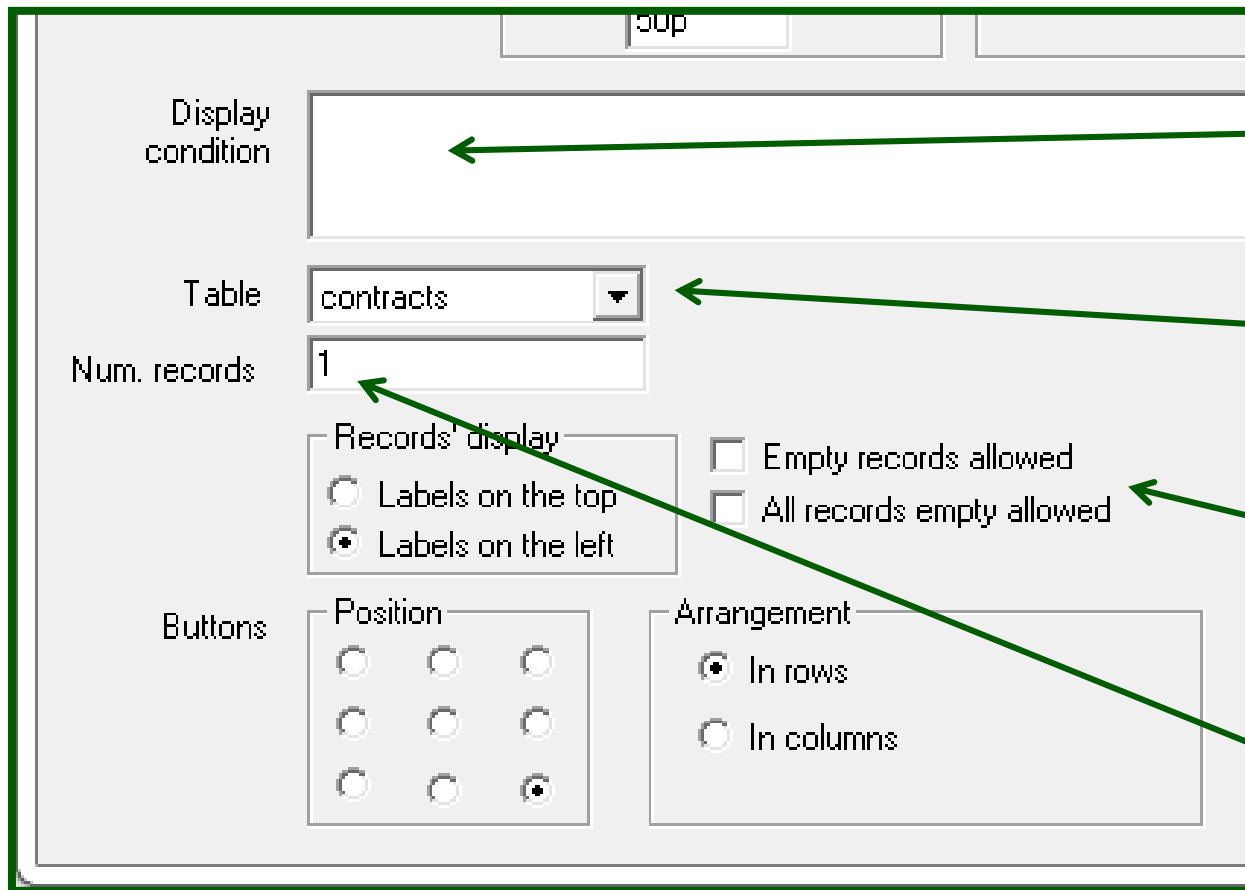
Contract list box: 

- Contracts can be selected
- Button in contract list box:
 - Program is executed for selected contract
 - Error message if no contract selected
 - Current subject's record in subjects table can be accessed using scope operator, e.g.

```
Creator = :Subject;
```

Box types

- Contract creation box: 
- Used to create new records in a z-Tree table:



The screenshot shows the 'Contract creation box' interface. It includes a 'Display condition' field, a 'Table' dropdown menu set to 'contracts', a 'Num. records' input field set to '1', and a 'Records' display section with radio buttons for 'Labels on the top' and 'Labels on the left' (selected). There are also checkboxes for 'Empty records allowed' and 'All records empty allowed'. The 'Buttons' section contains a 'Position' grid and an 'Arrangement' section with radio buttons for 'In rows' and 'In columns' (selected). Annotations with arrows point to the 'Display condition' field, the 'Table' dropdown, the 'Num. records' input, the 'Records' display section, and the 'Arrangement' section.

Condition for displaying box

Table wherein new records should be created

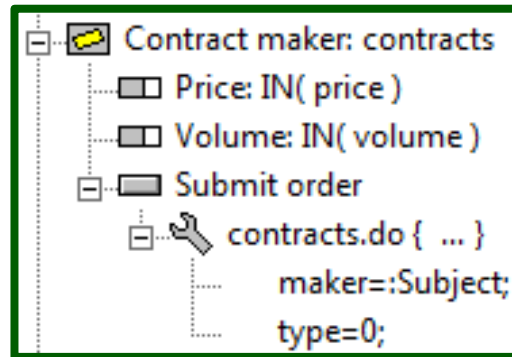
Additional settings for contract creation

Number of contracts which can be entered simultaneously

Box types

Contract creation box:

- Code:



- Output:



Price	23
Volume	5
Submit order	

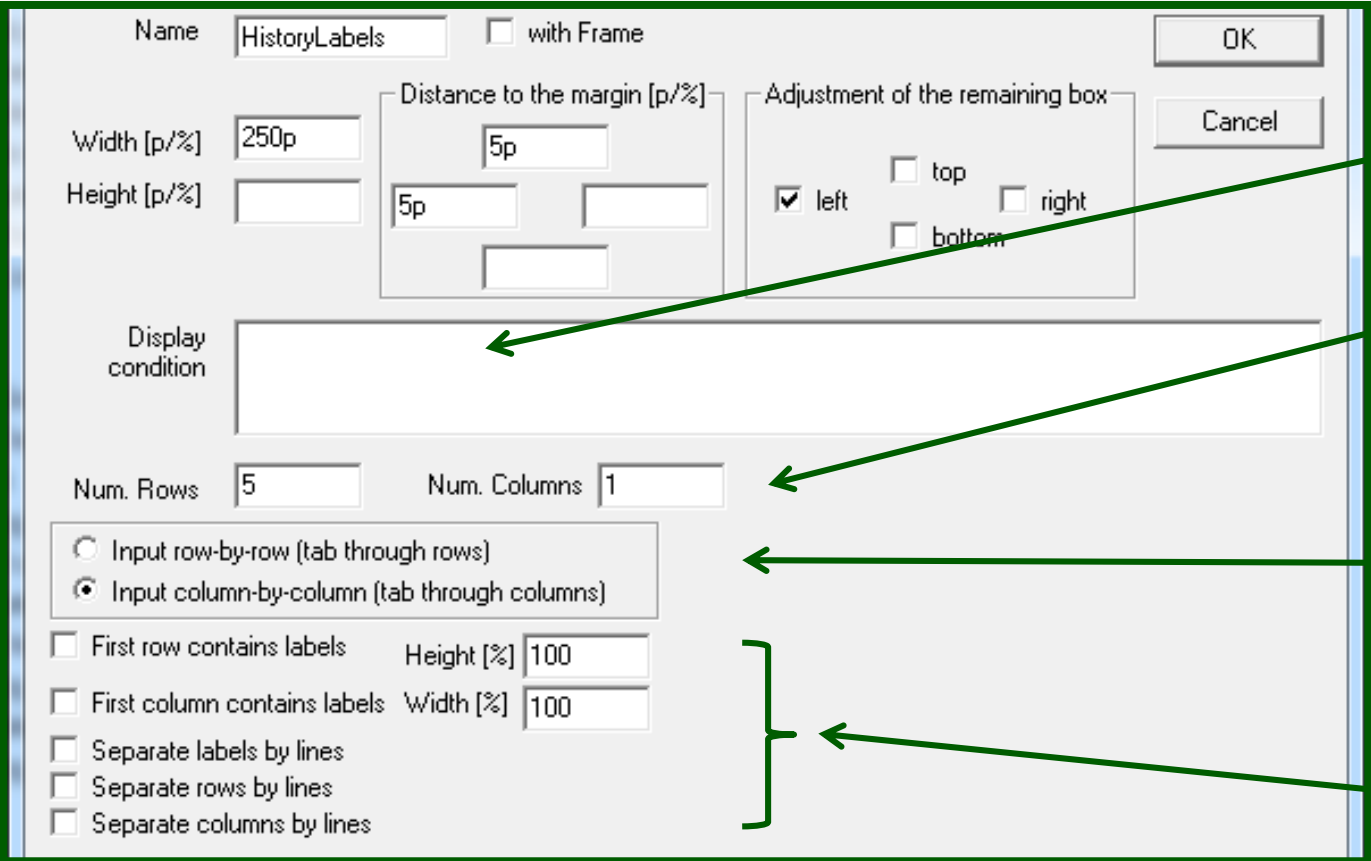
Contract creation box: 

- Item entry limitations determine what values are permissible
- Program in button
 - Used to write additional variables
 - Automatically limits its operation to new record

Box types

Grid box: 

- Used to display records from a z-Tree table:



The screenshot shows a dialog box titled 'HistoryLabels' with various configuration options. Green arrows point from text boxes on the right to specific fields in the dialog:

- Condition for displaying box:** Points to the 'Display condition' text area.
- Defines size of the box:** Points to the 'Width [p/%]' field, which is set to '250p'.
- Determines how the ordering of the items is interpreted:** Points to the 'Num. Rows' field, which is set to '5'.
- Formatting options:** Points to a group of checkboxes including 'First row contains labels', 'First column contains labels', 'Separate labels by lines', 'Separate rows by lines', and 'Separate columns by lines'.

Other visible fields in the dialog include 'Name' (HistoryLabels), 'with Frame' (unchecked), 'Distance to the margin [p/%]' (5p), 'Adjustment of the remaining box' (left checked, top, right, bottom unchecked), 'Height [p/%]' (empty), 'Num. Columns' (1), and radio buttons for 'Input row-by-row' and 'Input column-by-column' (selected).

Box types

Grid box: 

- Code: 

Attention:
Drawing a grid box with many items takes z-Tree a long time. Whenever possible, use e.g. a contract list box instead.

- Output:



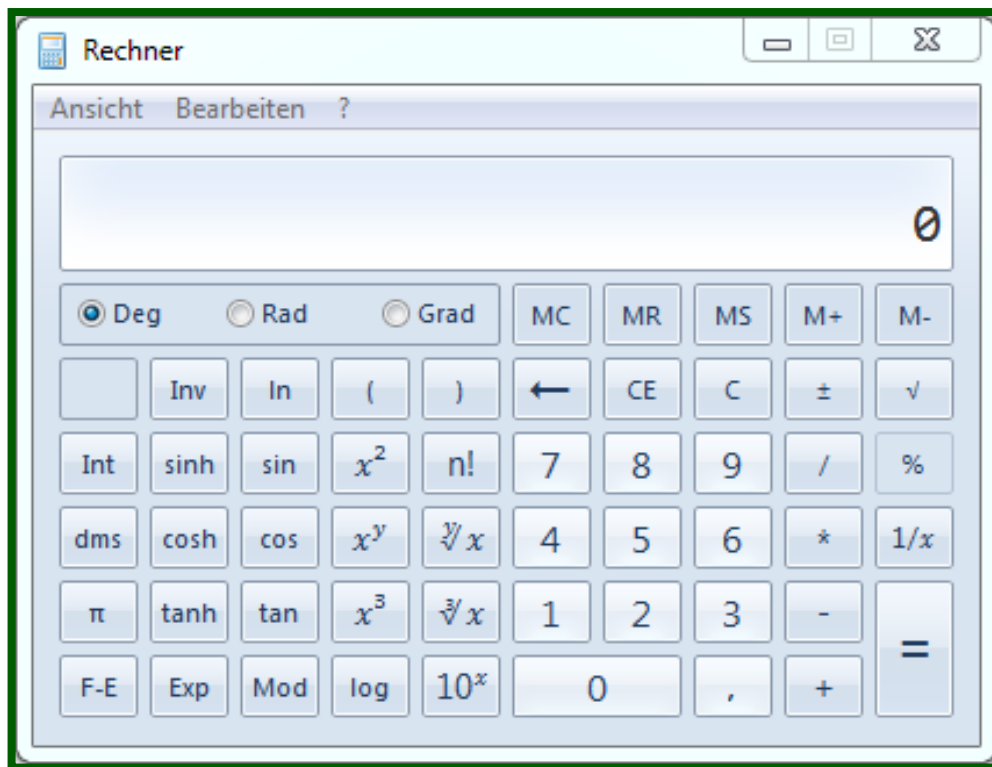
Gruppe mit Beitragswahl durch zufällig bestimmten Spieler

Periode	1	2	3	4	5	6	7	8
Anzahl Spieler	1	1						
Durchschnittlicher Beitrag/Kopf	15.00	15.00						
Durchschnittlicher Ertrag/Kopf	9.00	9.00						
Durchschnittlicher Endpunktestand	14.00	14.00						

Calculator button box

Calculator buttonbox: 

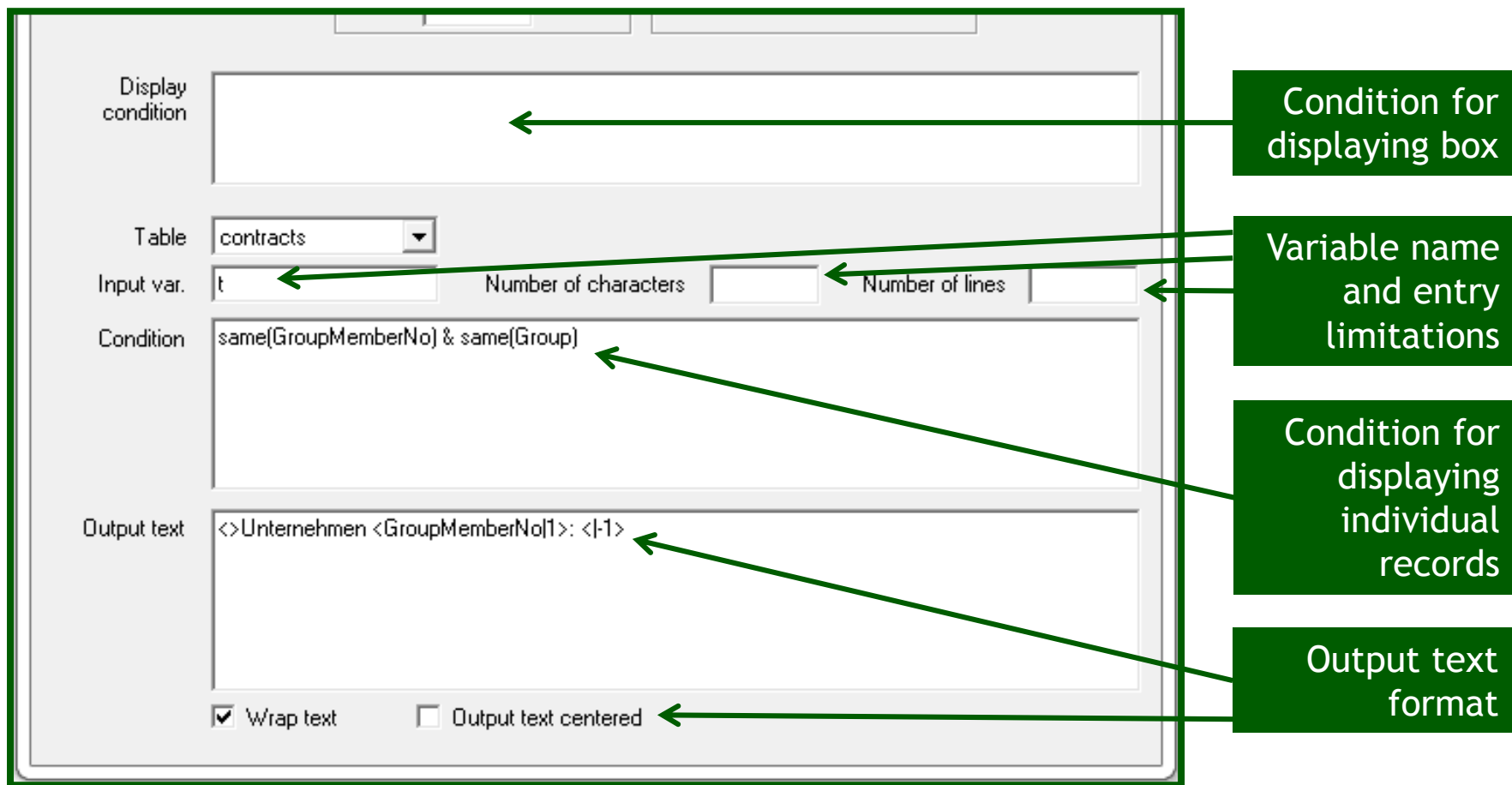
- Gives subjects access to the Windows calculator



Box types

Chat box: 

- Used to display/exchange chat messages:



The screenshot shows the configuration window for a Chat box. It includes several fields and checkboxes, with green arrows pointing from descriptive text boxes on the right to specific elements in the window.

Field	Value
Display condition	
Table	contracts
Input var.	t
Number of characters	
Number of lines	
Condition	same(GroupMemberNo) & same(Group)
Output text	<>Unternehmen <GroupMemberNo[1]>: < -1>
Wrap text	☒
Output text centered	☐

Annotations (from right to left):

- Condition for displaying box (points to Display condition)
- Variable name and entry limitations (points to Input var. and Number of lines)
- Condition for displaying individual records (points to Condition)
- Output text format (points to Output text)

Box types

Chat box: 

- Code:

```
OwnMessages: IN( t ), contracts( same(GroupMemberNo) & same(Group), <> Unternehmen <GroupMemberNo|1>: <|-1> )
├── contracts.do { ... }
│   ├── Subject=:Subject;
│   ├── GroupMemberNo=:GroupMemberNo;
│   └── Group=:Group;
└── OthersMessages: contracts( same(Group), <> Unternehmen <GroupMemberNo|1>: <t|-1> )
```



- Output:

```
Unternehmen 2: Hello
Unternehmen 1: This is a test!
Unternehmen 3: What is going on?
Unternehmen 1: What do you mean?
```

```
Unternehmen 1: This is a test!
Unternehmen 1: What do you mean?
```

Subject Progression

- Periods and stages in z-Tree
- Stage options
- Controlling subject progression
 - Subjects in different stages
 - Subjects in different “periods”
 - Random ending period

- Periods in z-Tree
 - All subjects in same period
 - Practice periods
- Stages in z-Tree
 - Subjects can be in different stages
 - Stages can require all subjects to enter jointly
 - Stages can be limited to one subject at a time
 - Stages can be skipped
 - Ending conditions
- Display conditions
 - The same stage can look different to different subjects

Stage options

- Start conditions
 - All subjects enter at same time
 - Every subject enters when it reaches stage
 - Subjects enter by condition
 - Number of subjects in stage limited to 1 per group
- Timeout
 - If no input required
 - Forced exit
 - No strict timeout

The screenshot shows the 'Stage' dialog box in the z-Tree software. The dialog has a title bar with a close button (X). It contains the following fields and options:

- Name:** A text field containing the word 'Stage'.
- Start:** A section with three radio button options:
 - ☐ Wait for all
 - ☐ Start if possible
 - ☒ Start if... (This option is selected, and there is an empty text field next to it for specifying a condition.)
- Number of subjects in Stage:** A section with a checkbox labeled 'At most one per group in stage', which is currently unchecked.
- Leave stage after timeout:** A section with three radio button options:
 - ☒ If no input
 - ☐ Yes
 - ☐ No
- Timeout:** A text field containing the number '30'.
- Buttons:** 'OK' and 'Cancel' buttons are located on the right side of the dialog.

Controlling subject progression

- Simulation of stages through boxes and display conditions
- Participate variable in subjects table
 - Set to 0 to make a subject skip this stage (must be done in the stage background)
- Priority variable in subjects table
 - Determines order in which subjects enter stage when “At most one per group in stage” is set in a stage
 - Subject with smallest Priority enters first
 - Subjects enter randomly if Priority not set

Controlling subject progression

- LeaveStage variable in subjects table
 - Set to 1 to make a subject move immediately to the waiting screen of the current stage
- RepeatTreatment variable in globals table
 - Checked after last period
 - Set to >0 to repeat the treatment with the same number of periods (ideal for random ending period)
 - Period number increments as if it was all one treatment
 - Allows for random ending feature with external randomization device

Optimized progression within a period

- Different roles have different tasks, or different groups progress at different speeds
- Some require prior input by other roles, some do not
- Solution:
 - Use “Wait for all” where all roles’ input required
 - Use “Start if...” where some conditions have to be met

Exercise: Random ending period

Link to
Random ending period
in
Exercises

Optimized progression within a period

- Things to consider:
 - Is it better for subjects to wait a little at multiple points vs. a lot at one point?
 - Make sure all variables are defined in background
 - Check rules for running programs in table definition
 - Account for subjects' progress when making calculations for them
 - Allow sufficient time for programming and testing

Exercise: Progression within a period

Link to
Progression within a period
in
Exercises

Simulating subjects in different periods

- Program everything in a single stage, using stacked containers/boxes
- Switch between different screens using display conditions
- Use user-generated period variable (e.g. SubjectivePeriod)
- Use user-generated header
- Put programs into buttons instead of stage backgrounds

Import and Export

- Parameter table import
- Table dumper and table loader

Parameter table variable import

- Variables can be imported from ASCII file (one-by-one)
- Allows parameter customization before session

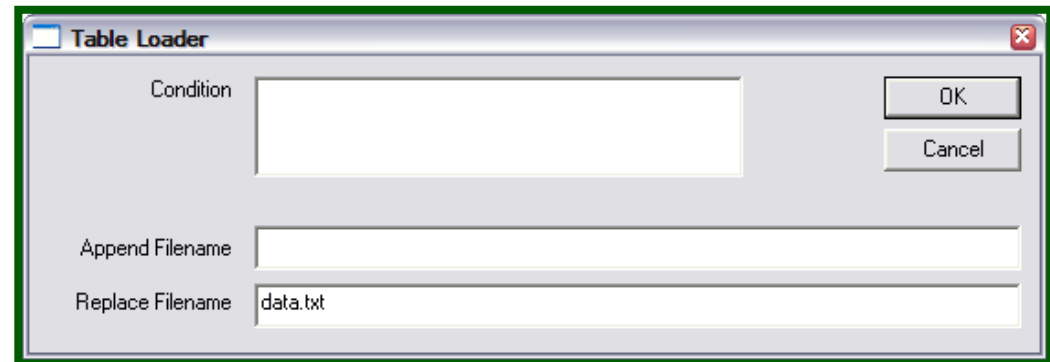
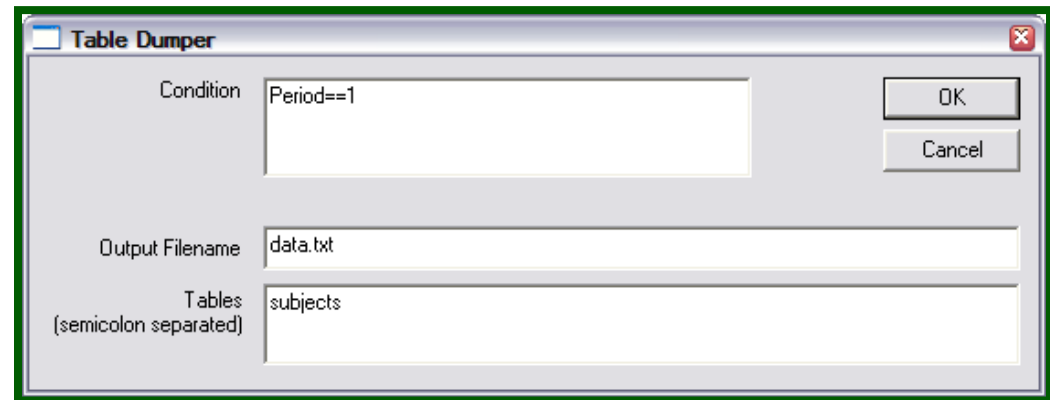
The first screenshot shows the zTree interface with the 'Import Variable Table...' option selected in the 'Tools' menu. A green arrow points from this menu item to the 'Specific Parameter' dialog box in the second screenshot. The dialog box shows 'Subject S 2', 'Period 1', 'Group 1', and a 'Program' field containing the code: `cash = 225+loan;` and `stock = 3;`. Another green arrow points from the 'Program' field to the third screenshot, which shows a text editor window titled 'parameters2_12subjects_cash.txt'. The editor displays the following tab-separated values:

Datei	Bearbeiten	Format	Ansicht	
225+loan	225+loan	225+loan	225+loan	
945+loan	945+loan	945+loan	945+loan	
585+loan	585+loan	585+loan	585+loan	

Green arrows point from the first three columns of the table to a green box labeled 'Tab-separated values'.

Data import and export 1/3

- Tables can be imported and exported from/to ASCII file
 - Table dumper:
Allows exporting table to ASCII file (Mac: Windows (CRLF), UTF-8)
 - Table loader:
Allows importing table from ASCII file (append or replace)



- Demo_DataFileRW.ztt
- Table dumper:

parameters	Period	x
parameters	1	68
parameters	1	3
parameters	1	94

- Table loader (replace): Replaces table records from first downwards with data in ASCII file
- Table loader (append): Adds new table records after last one

Exercise: Data import and export

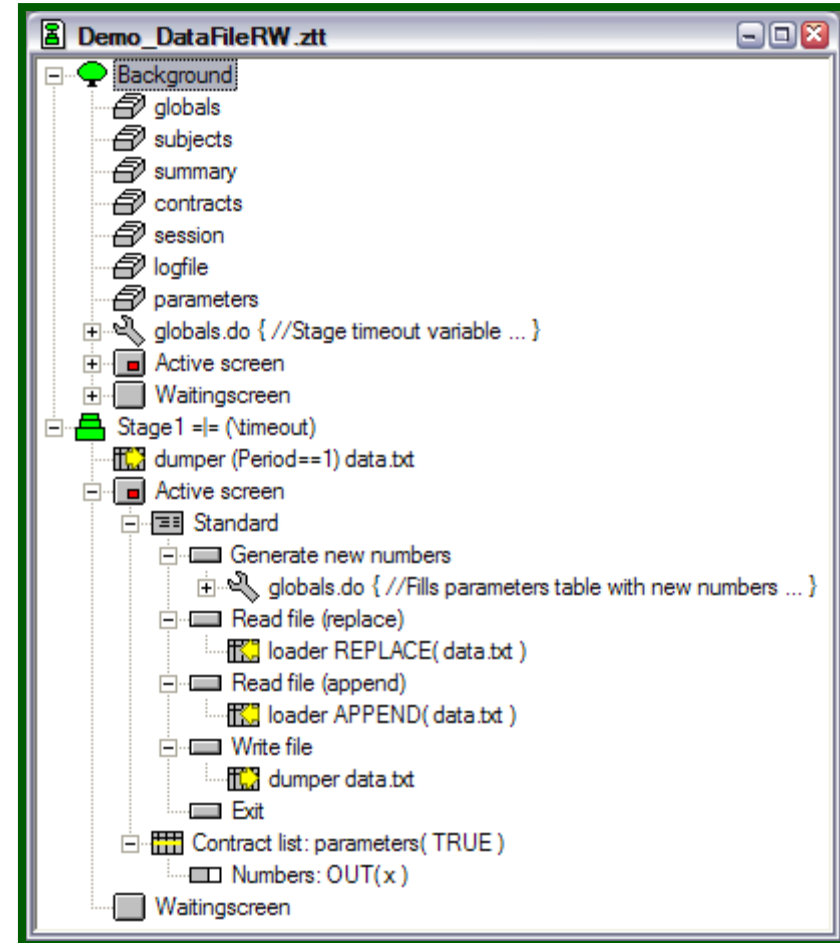
- Program Demo_DataFileRW.ztt

Data import and export 3/3

- Demo_DataFileRW.ztt
- Table dumper:

parameters	Period	x
parameters	1	68
parameters	1	3
parameters	1	94

- Table loader (replace):
Replaces table records
from first downwards with
data in ASCII file
- Table loader (append):
Adds new table records
after last one



Link to
Generating repetitive code with Excel
in
Good Practice and Advice

Questionnaires

- Overview
- Using the session table
- Checking answers
- Presenting results

- Run after ≥ 1 treatment (sets number of subjects)
- Address form [sic!] needed to write payment file
(delete components by deleting caption in address form)
- Question forms used for:
 - Solicitation of feedback
 - Display of results
- Questionnaire may be empty

Overview

- Rules set the regions where labels and questions are positioned
- Maximum one button per question form
- Last question form may not contain a button

Adresse

Socioeconomics: Background data

l=9%; r=9%; s=7p; label=61%

☐ Your Gender?: IN(Female)

☐ How old are you?: IN(Age)

☐ Nationality: IN(Nationality)

Continue

Profit:

Your earnings from this experiment are:: OUT(MoneyEarned)

Thank you for your participation !

Using the session table

- Conditioning on variables in session table (Participate)
- Run treatment writing variables into session table (FinalProfit, MoneyAdded, ShowUpFee, MoneyToPay and MoneyEarned are always available)
- Access variables in session table in question form definition in questionnaire
- Use Participate variable to control which subjects see which parts of the questionnaire

Using the session table

- Example: Use a variable x from the subjects table.
- In the treatment create a program in the session table:
 - $x = : x;$
- In the questionnaire, use a text like:
 - $\langle \rangle$ Displays x : $\langle x \mid 1 \rangle$

Checking answers

- Automatic checking of control questions
- Use normal treatment (i.e. not a questionnaire)
- Use buttons with checkers

Link to
Presenting questionnaire results
in
Managing z-Tree Output

Managing z-Tree Output

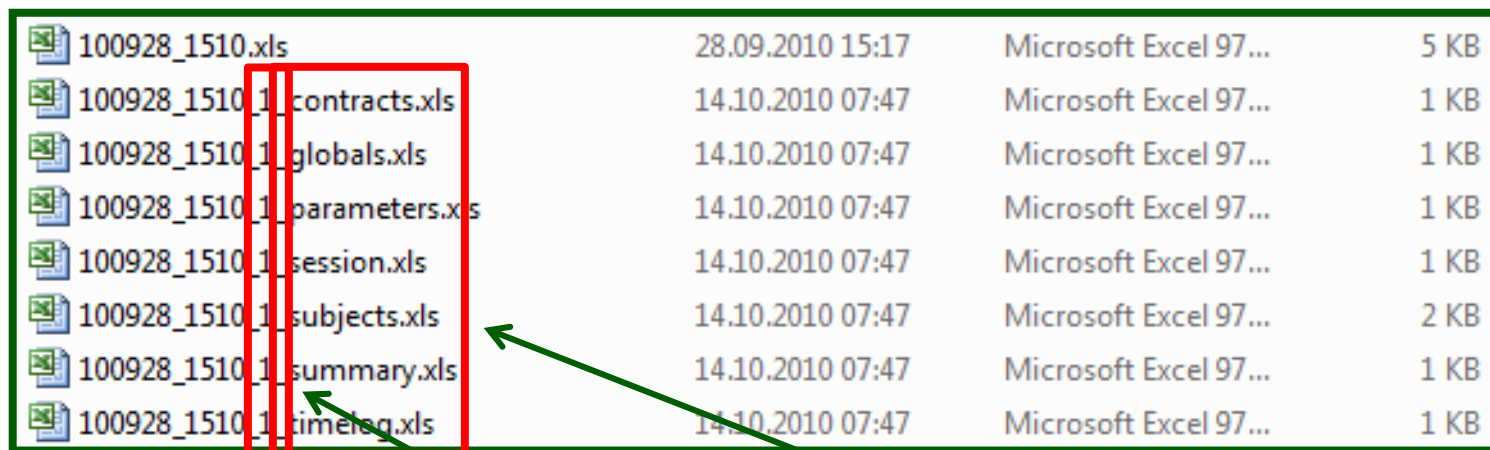
- Reformatting z-Tree output
- Presenting questionnaire results
- Import into R
- Import into and processing in Stata
- Results table

Reformatting z-Tree output

- Main data is saved in YYMMDD_hhmm.xls
- Reformatting possibilities:
 - Manually
 - Using z-Tree's "Tools - Separate tables..." command (see following slides)
 - Using my Excel add-in zTools (see following slides; obtain from <http://academic.palan.biz/downloads/ztools>)
 - Using Kan Takeuchi's Stata import procedure
 - Using Oliver Kirchkamp's R import procedure

Reformatting z-Tree output

- z-Tree's “Tools - Separate tables...” command
 - Select .xls-file to process
 - Output:

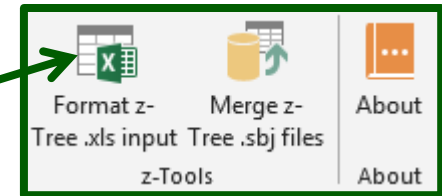


100928_1510.xls	28.09.2010 15:17	Microsoft Excel 97...	5 KB
100928_1510_1_contracts.xls	14.10.2010 07:47	Microsoft Excel 97...	1 KB
100928_1510_1_globals.xls	14.10.2010 07:47	Microsoft Excel 97...	1 KB
100928_1510_1_parameters.xls	14.10.2010 07:47	Microsoft Excel 97...	1 KB
100928_1510_1_session.xls	14.10.2010 07:47	Microsoft Excel 97...	1 KB
100928_1510_1_subjects.xls	14.10.2010 07:47	Microsoft Excel 97...	2 KB
100928_1510_1_summary.xls	14.10.2010 07:47	Microsoft Excel 97...	1 KB
100928_1510_1_timelog.xls	14.10.2010 07:47	Microsoft Excel 97...	1 KB

- Separated by treatments and tables

Reformatting z-Tree output

- Using zTools
 - Click “Format z-Tree .xls input”
 - Answer dialog questions
 - Output (new Excel file):

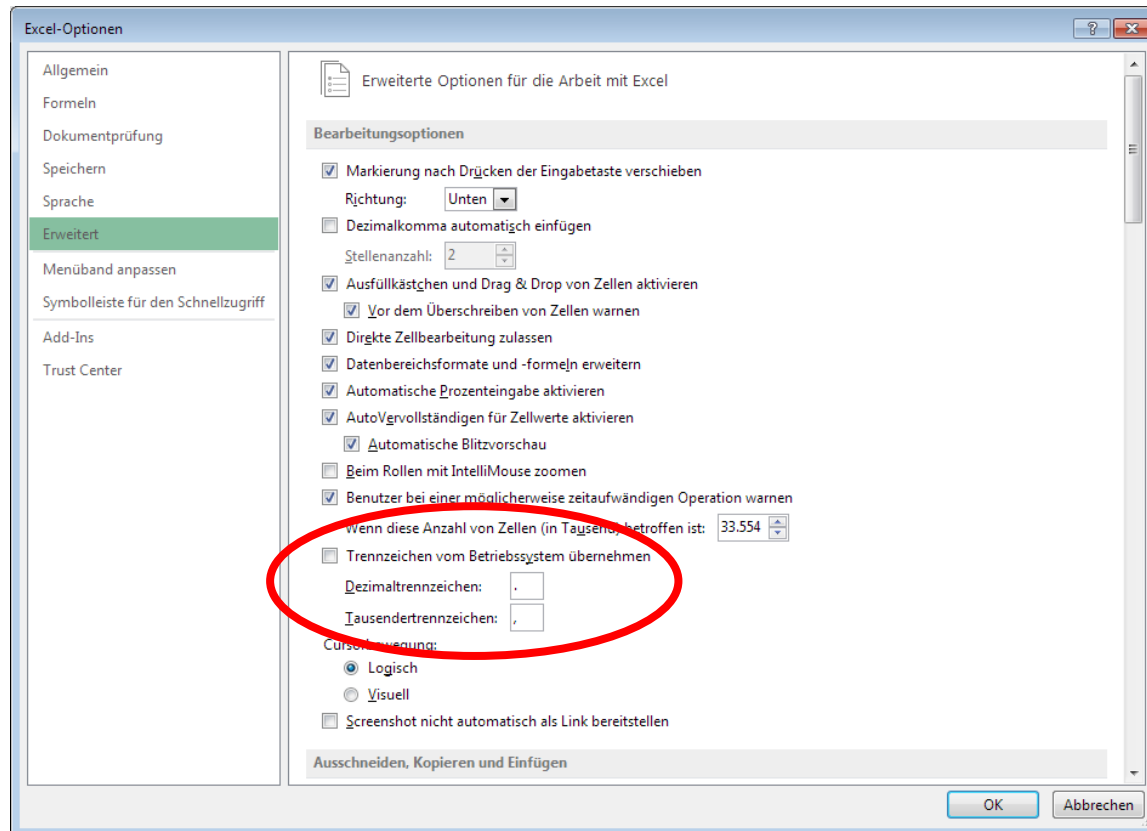


The screenshot shows an Excel spreadsheet with a table of data. The columns are labeled A through H. The data is organized into rows, with the first row (row 1) containing headers: '100304_1001', '1 subjects', 'Period', 'Subject', 'Group', 'Profit', and 'TotalProf'. The subsequent rows (rows 2-16) contain numerical data for each of these categories. The 'subjects' column shows values from 1 to 11, then 2, 2, 3, 4. The 'Period' column shows values 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2. The 'Subject' column shows values 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1, 2, 3, 4. The 'Group' column shows values 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1. The 'Profit' column shows values 188.727273, 171.818182, 13.4545455, 231.818182, 85.7272727, 62.7272727, 81.9090909, 69.0909091, 82.3636364, 105.090909, 0, 1.81818182, 190.5454545, 48.0909091, 219.9090909, 102.727273, 116.181818, -21.8181818. The 'TotalProf' column shows values 188.727273, 171.818182, 13.4545455, 231.818182, 85.7272727, 62.7272727, 81.9090909, 69.0909091, 82.3636364, 105.090909, 0, 1.81818182, 190.5454545, 48.0909091, 219.9090909, 102.727273, 116.181818, -21.8181818. The 'RawData' tab is selected, and the 'subjects' column is highlighted with a red oval.

	A	B	C	D	E	F	G	H
1	100304_1001	1 subjects	Period	Subject	Group	Profit	TotalProf	
2	100304_1001	2 subjects	1	1	1	188.727273	188.727273	
3	100304_1001	2 subjects	1	2	1	171.818182	171.818182	
4	100304_1001	2 subjects	1	3	1	13.4545455	13.4545455	
5	100304_1001	2 subjects	1	4	1	231.818182	231.818182	
6	100304_1001	2 subjects	1	5	1	85.7272727	85.7272727	
7	100304_1001	2 subjects	1	6	1	62.7272727	62.7272727	
8	100304_1001	2 subjects	1	7	1	81.9090909	81.9090909	
9	100304_1001	2 subjects	1	8	1	69.0909091	69.0909091	
10	100304_1001	2 subjects	1	9	1	82.3636364	82.3636364	
11	100304_1001	2 subjects	1	10	1	105.090909	105.090909	
12	100304_1001	2 subjects	1	11	1	0		
13	100304_1001	2 subjects	2	1	1	1.81818182	190.5454545	
14	100304_1001	2 subjects	2	2	1	48.0909091	219.9090909	
15	100304_1001	2 subjects	2	3	1	102.727273	116.181818	
16	100304_1001	2 subjects	2	4	1	-21.8181818		

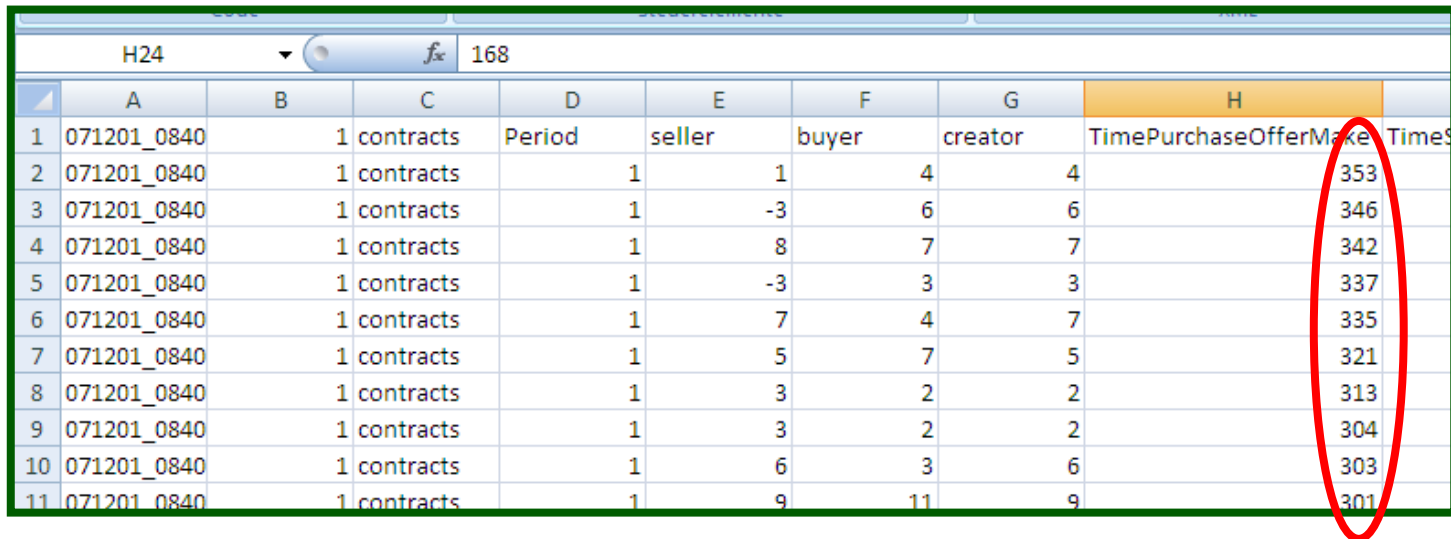
Topics in data processing

- Opening .xls output file in German Excel creates problems with data interpretation (decimals interpreted as dates)
- Solution:



Topics in data processing

- Time data
 - Recorded for all button clicks (if stage has a timeout)
 - Times are recorded as time remaining in period:



The screenshot shows a z-Tree experiment interface. At the top, there is a dropdown menu set to 'H24' and a text box containing '168'. Below this is a table with 11 rows and 10 columns. The columns are labeled A through H. Column A contains a sequence of IDs (071201_0840). Column B contains the number '1'. Column C contains the word 'contracts'. Column D is labeled 'Period' and contains the number '1'. Column E is labeled 'seller' and contains the number '1'. Column F is labeled 'buyer' and contains values 1, -3, 8, -3, 7, 5, 3, 3, 6, 9. Column G is labeled 'creator' and contains values 4, 6, 7, 3, 4, 7, 2, 2, 3, 11. Column H is labeled 'TimePurchaseOfferMake' and contains values 353, 346, 342, 337, 335, 321, 313, 304, 303, 301. A red oval is drawn around the 'Time' column header and its corresponding values.

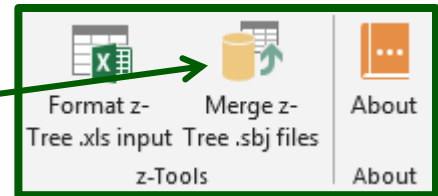
	A	B	C	D	E	F	G	H	
1	071201_0840	1	contracts	Period	seller	buyer	creator	TimePurchaseOfferMake	TimeS
2	071201_0840	1	contracts	1	1	1	4	4	353
3	071201_0840	1	contracts	1	-3	6	6	6	346
4	071201_0840	1	contracts	1	8	7	7	7	342
5	071201_0840	1	contracts	1	-3	3	3	3	337
6	071201_0840	1	contracts	1	7	4	7	7	335
7	071201_0840	1	contracts	1	5	7	5	5	321
8	071201_0840	1	contracts	1	3	2	2	2	313
9	071201_0840	1	contracts	1	3	2	2	2	304
10	071201_0840	1	contracts	1	6	3	6	6	303
11	071201_0840	1	contracts	1	9	11	9	9	301

Presenting questionnaire results

- Manually
 - Transpose and copy .sbj file contents to Excel sheet
 - Make sure to account for extra text columns
 - Put data under correct headings
- Using zTools (see next slide)
- Add session and treatment identifiers

Presenting questionnaire results

- Using my Excel add-in zTools
 - Click “Merge z-Tree .sbj files”
 - Select questionnaires to merge
 - Output (in same Excel file):



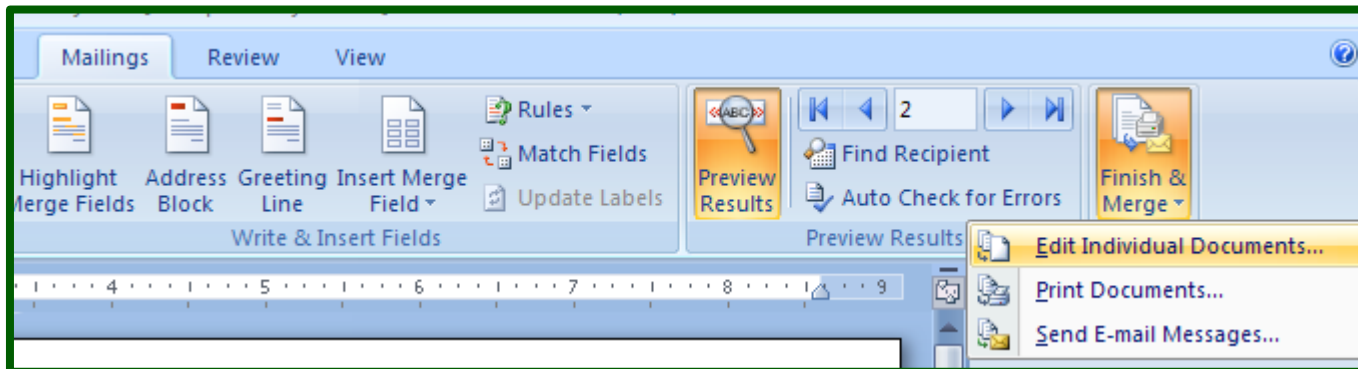
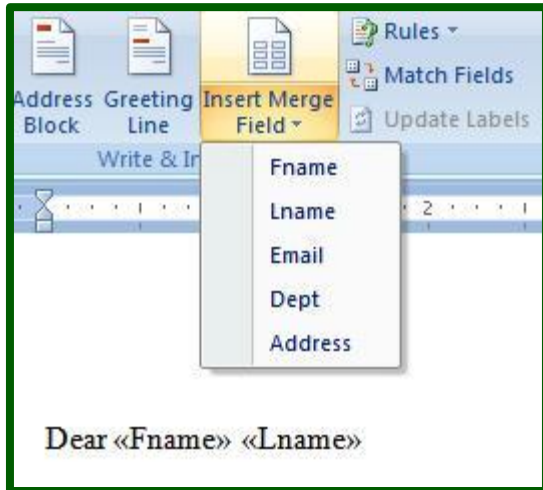
The screenshot shows an Excel spreadsheet with the following data:

	A	B	C	D	E	F
1	Sheets	Subjects	Variables			
2	131002_1328	13	18			
3	131205_0824	13	19			
4						
5						

The spreadsheet also shows the zTools ribbon at the top with the 'Merge z-Tree .sbj files' button highlighted. The bottom status bar indicates 'BEREIT' and '100%' zoom.

Presenting questionnaire results

- Use Word mailings function for formatting and printing or .pdf output



Presenting questionnaire results

«SessionID»—Subject:«Subject»

→

(«client»)¶

«Nationality», «Female», «Age», «Non-smoker-(«NumberCigarettes»», «Major»-
«LevelDegree»-(«YearsStudy»st-year), «Employment»:«Employment»¶

Monthly-household-income-before-taxes:«HouseholdIncome»¶

Extreme-Sports:«Extreme S19M36-US15-B—Subject:1

→

(05-Tyran5040)¶

What-was-your-strategy-d
«strategy»«strategy_1»«strategy_2»¶

Eritrean, «male», «26», «Non-smoker-(0)», «Compute-science-Master-(1st-year)», «Employment»:none¶

Monthly-household-income-before-taxes:5.000-kr--10.000-kr¶

Was-it-difficult-for-you-to-d
Extreme-Sports:Never¶

What-was-your-s
Selling-with-high-price-in-the-market-experiment?¶

Do-you-believe-that-you-ac

Did-you-ever-make-a-mista
us-exactly-what-went-wron
«typo»«typo_1»«typo_2»¶

Was-it-difficult-f
e-beginning-(1-7):5¶

Do-you-believe-t
(1-7):3¶

Estimated-rank-(1-10):«ra

Did-you-find-the-instruction
anything-could-be-improve
«clarityinstructionsmarket»«clar

Did-you-ever-make-a-mistake-in-entering-a-price, or-clicked-a-wrong-button? If-so, please-tell-
us-exactly-what-went-wrong-and-in-what-period:¶

I-made-a-mistake-entering-the-price-in-the-first, but-not-clicking-the-wrong-button¶

Estimated-rank-(1-10):-2-(Real-rank:7)¶

How-did-you-make-your-ch
«strategyHL»«strategyHL_1»¶

Did-you-find-the-instructions-in-the-market-experiment-clear-and-understandable? What-if-
anything-could-be-improved?¶

for-me, i-would-need-some-tim-completely-understanding-the-instructions, i-figure-out-most-of-them-in-the-process¶

Did-you-find-the-instruction
What-if-anything-could-be-
«clarityinstructionsHL»«clarityins

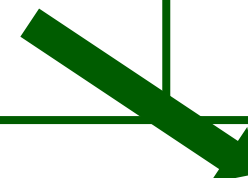
How-did-you-make-your-choice-in-the-lottery-choice-experiment?¶

Room-for-your-comments:«
«comments»«comments1»¶

Did-you-find-the-instructions-in-the-lottery-choice-experiment-clear-and-understandable? What-if-
anything-could-be-improved?¶

Room-for-your-comments:¶
0¶

Return to
Questionnaires



- Use Oliver Kirchkamp's zTreeTables command:

Importing z-Tree tables in R

For those who use R, here is a utility to import data from z-Tree into R. It provides one command:

- `zTreeTables (filelist, tables=c("globals", "subjects"))`

The return value is then a list of all merged tables.

Example:

```
source("http://www.kirchkamp.de/lab/zTree.R")
zTT <- zTreeTables(c("080712_1412.xls", "080712_1818.xls"))
with(zTT$subjects, table(Treatment, Period))
```

- Use Kan Takeuchi's `ztree2stata` command:

`ztree2stata.ado`

Kan Takeuchi*

March 01, 2005

Installation: copy `ztree2stata.ado` and `ztree2stata.hlp` into `C:/stata8/ado/base/z`. If you cannot find such a directory, then run `sysdir` command and see [U] **20.7 How do I add my own ado-files?**.

Syntax

`ztree2stata` *table* using *filename* [, treatment(*numlist*) string(*string_varlist*) except(*string*)
save replace clear]

Description

`ztree2stata` imports a z-Tree (Fischbacher, 1999) data file and converts it into Stata format. Specifically `ztree2stata`:

1. opens an MS-Excel file saved by z-Tree,
2. deletes all data other than that of the specified table by *table*,
3. renames variables using the variable names in z-Tree, and
4. converts the data into numerical type if it is not in the *string_varlist*.

Restructuring and formatting data in Stata

- describe
- list
 - list in 1/8
- drop/keep
 - drop varA varB
 - keep if varA==5
- rename
 - ren varA varC
- recode
 - recode varA 5 7.25 = 13

Restructuring and formatting data in Stata

- generate/replace
 - `gen varD=2*varB`
 - `replace varD=varD+1 if varB==6`
- egen
 - `egen varE=mean(varA)`
 - `egen varD=mean(varA) if a==4, by(varD)`
- collapse
 - `collapse (mean) MeanPrice=Price Cash (median) MedianPrice=Price, by(Period)`
- reshape
 - See help...

- summarize
 - sum varA varB
 - sum varA varB, detail
- histogram
 - hist varA
- tabulate twoway (tab2)
 - tab2 varA varB
- ttest
 - ttest varA==varB
 - ttest varA==5 if varB==3
- regress/probit/logit
 - reg varA varB varC

Results table

- There are always many ways to prepare your results for analysis
- Use pre-defined tables where available
 - Per-period data: use summary table
 - Per-subject data: use subjects table
 - You can never have too much information in an output table (i.e. do not create a user-defined table just to copy a subset of e.g. the subjects table into it)

- Know the strengths of z-Tree and of your statistics package
 - Only perform transformations and calculations in z-Tree that are easier there
 - Easy to combine information from different tables or different subjects into one table row in z-Tree
 - Easy (and fast!) to calculate new variables from the information in a single table row in e.g. Stata
- Examples:
 - Use z-Tree to copy responder data to proposer row
 - Calculate mean, variance of subjects' earnings in Stata

Troubleshooting and Ensuring You Don't Need to

- Preparing for the experiment
- Debugging
- Crash handling

Session structure help sheet

- Prepare session structure help sheet (“script”)
 - Login data for lab
 - Folders and technical details
 - Program settings
 - ...

General Instructions

General

Run ztree.exe (or restart ztree.exe if this is not the first market run – otherwise profits are pooled over sessions). Set process priority to Higher for ztree.exe (Task Manager).

Connect zleaf clients.

Background

Set number of players=11 (10 participants + 1 experimenter) in treatment and Holt/Laury.

Program in globals table “### Global parameters”

Set TimeAuction=240.

Set TimeProfit=30.

Set experimenters=1.

Set exchangerate correctly.

Set testing=0.

Set runfinlit=0.

Set payofffinlit correctly (set to zero if no payout wanted).

Set nospec=1 (no speculation treatment) or =0 (speculation treatment).

Set practiceperiod=0.

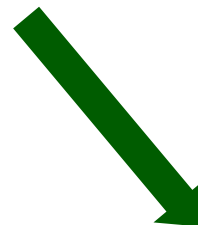
Commenting

- Comment your code extensively:

- Single-line commenting:
// until the end of the line

- Inline commenting:
x /* comment */ =5;

- Multi-line
commenting:
/* multi
line
comment */



```
//Reduces liquidity and cash of old offer if it is <= liquidity of new offer
if (liquidity<=:liquidity)
{
  //Switches current contract inactive, so its liquidity is not reduced in the control run
  //below
  activecontract=0;
  //Reduces cash of both subjects involved and adjusts open offers (CDA and Digital)
  subjects.do
  {
    if (Subject==:offerer | Subject==:offerer)
    {
      kapital=kapital-liquidity;
      optioncosts=optioncosts+liquidity;
      //Adjusts Digital market open offers
      redoffers.do {if (:Subject==offerer & activecontract==1 & liquidity>:kapital) {liquidity=:kapital;}}
      //Adjusts CDA open offers
      contracts.do {if (:Subject==buyer & seller==1 & preis>:kapital) {seller=-3;}}
    }
  }
  //Switches current contract active again
  activecontract=1;
  //Adjusts liquidity of both offers
  :liquidity=:liquidity-liquidity;
  liquidity=0;
  activecontract=0;
}
```

- Test extensively
 - Ask others to test - $2n$ eyes see more than 2 (for $n > 1$)
 - Test multiple periods
 - Test with all previous and subsequent treatments
- Attempt analysis based on data generated by test run

- Keep a documentation of tables and variables:

Documentation GIMS v5.7

8

transactions

(Life: Period)

Note that in the case of Call Auctions, buy and sell offers are not directly matched with each other. For this reason, executed buy and sell offers trigger separate transactions. This implies that for example calculating the sum of all transaction volumes in the transactions table would yield a number twice as large as the number of shares actually traded.

Variable	Content
Market	Market ID.
Price	Offered price.
Volume	Transacted volume.
OfferID	Corresponds to ID in offers table.
AcceptanceID	ID of this transaction.
AcceptorID	Subject number of accepting subject in the Continuous Double Auction.
Time	Transaction time.

pricedetermination

(Life: Period)

Table used to determine equilibrium price in call auction market.

Variable	Content
Market	Market ID.
Price	Unique price derived from auctionoffers.price.
BuyVol	Total buy order volume at exactly this price, excluding market orders.
SellVol	Total sell order volume at exactly this price, excluding market orders.

- Errors when closing a program
 - Read the error message
 - z-Tree usually puts the cursor into the line causing the error
 - Use comments to locate the error
- Errors while the program is running
 - Test different combinations of parameters
 - Consult the tables to check intermediate results
 - Do so within z-Tree (stop the clock)
 - Use the .xls output file
 - Use table dumpers to see at what point program crashes

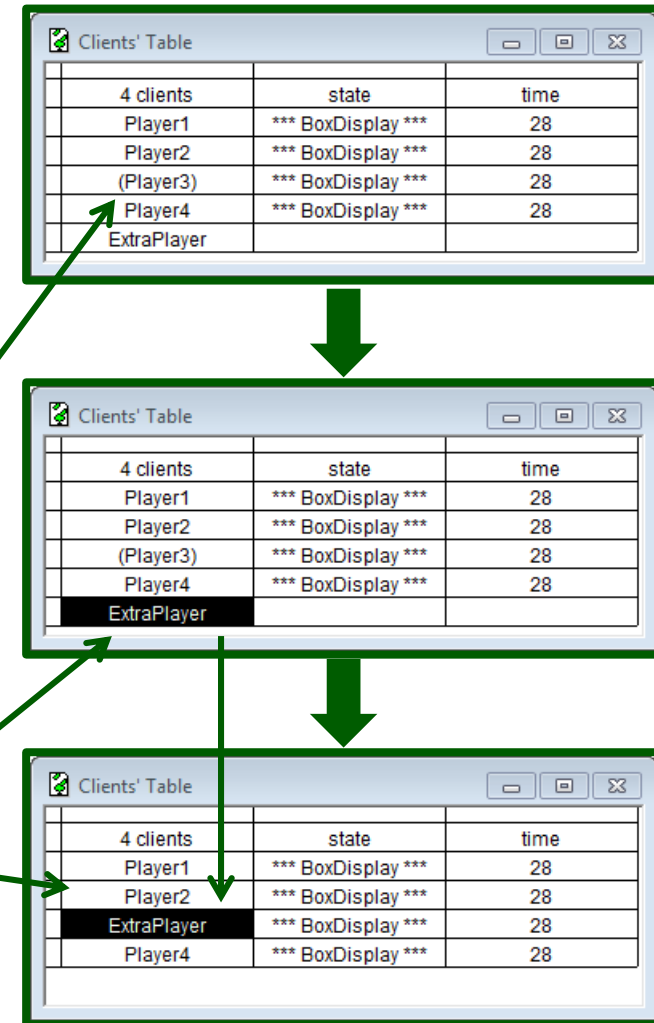
* Adapted from lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

- Client Crash
- If the PC is still working, simply restart the z-leaf (<Alt>-<F4> to shut it down)
- The z-leaf will then replay the treatment up to the present situation

* Adapted from lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

Crash handling*

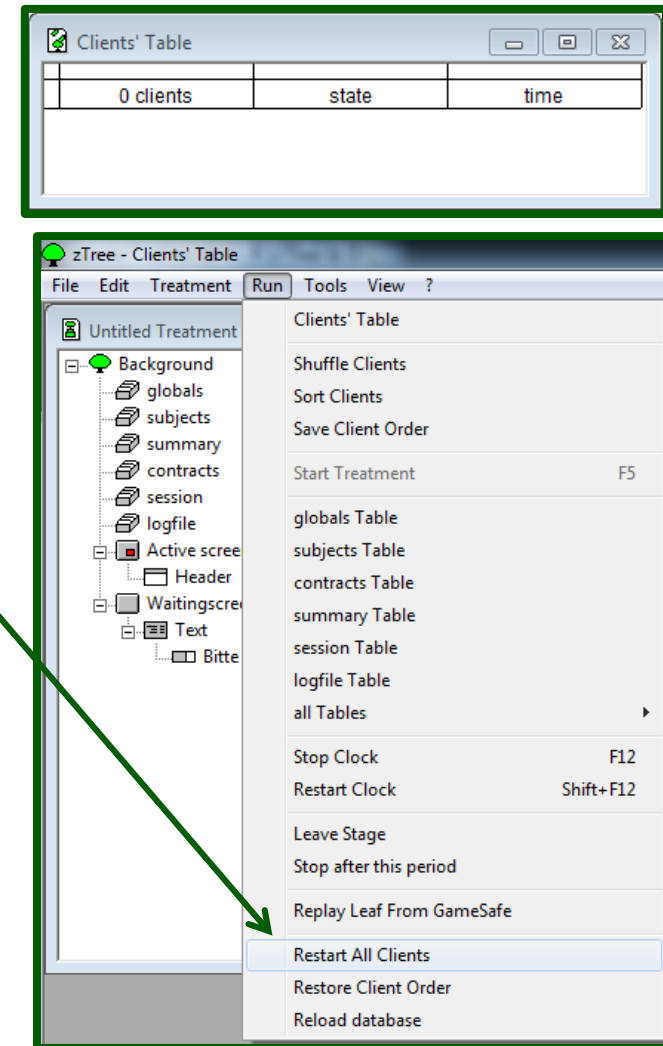
- Client Crash
- If the PC is not working, start a new PC
- Start z-leaf on new PC
- Open clients table on z-tree
- Disconnected client appears in parentheses
- Move new client over field of old client



* Adapted from lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

Crash handling*

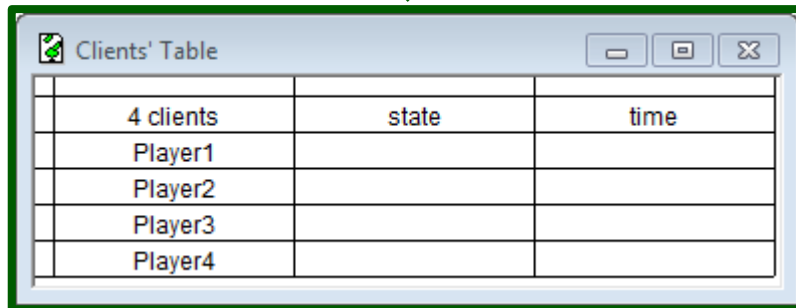
- Server Crash
- Restart z-Tree
- Open the clients' table
- Restart all clients with menu command “Restart all Clients”
- If some clients do not reconnect, manually restart their z-leafs



* Adapted from lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

Crash handling*

- Server Crash
- If no client connects, you can:
 - Restart the clients manually
 - Wait (up to 4 minutes) and restart the clients later
 - Start z-Tree on another computer
 - Shut down and restart the server PC



	4 clients	state	time
	Player1		
	Player2		
	Player3		
	Player4		

* Adapted from lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

Crash handling*

- Server Crash
- Use Run/Restore Client Order to sort the clients into the same order as before the crash
- Restore all tables with Run/Reload database (tables are stored after each period)
- Check the summary or subjects table to see how many periods have been played (n)
- Open the treatment that ran before the crash (e.g. @1.ztt)
- Set the number of practice periods to -n
- Restart treatment with Run/Start treatment (<F5>)

* Adapted from lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

Good Practice and Advice

- General advice
- Timelog
- Experimenter subject
- Calculating rank and sorting values
- Calculation of payouts
- Generating repetitive code with Excel
- Starting multiple z-Leafs for testing

General advice

- Program all treatments in one treatment file
- Use “speaking” variable names
- Define all variables in the Background
- Change timeout during the experiment*
 - Use a variable to set the timeout for a stage
 - Modify the variable in the period parameters in the parameter table (for periods yet to be played)
- „Stop after this period“ option*
 - Use both in testing and, if necessary, in an experiment
 - Period runs its course, but treatment then stops

* Source: z-Tree manual.

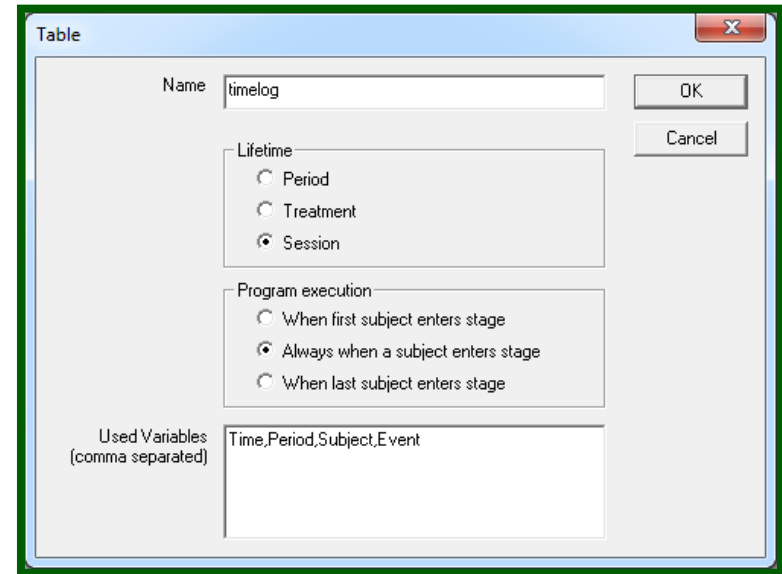
General advice

- Search and replace using export function
 - Use File-Export-Treatment to export code to a .txt-file.
 - Use search/replace in Notepad/Word...
 - Re-import treatment file into z-Tree
- Calculate exact time using `gettime ()`
 - `gettime()` returns time since computer was started
 - Set variable equal to `gettime()` at beginning of period
 - Calculate difference between time of interest and time calculated at the beginning

- Record exact timing of all events in the program:
- Calculate exact time using `gettime ()`
 - `gettime()` returns time since computer was started
 - Set variable equal to `gettime()` at beginning of period
 - Calculate difference between time of interest and time calculated at the beginning

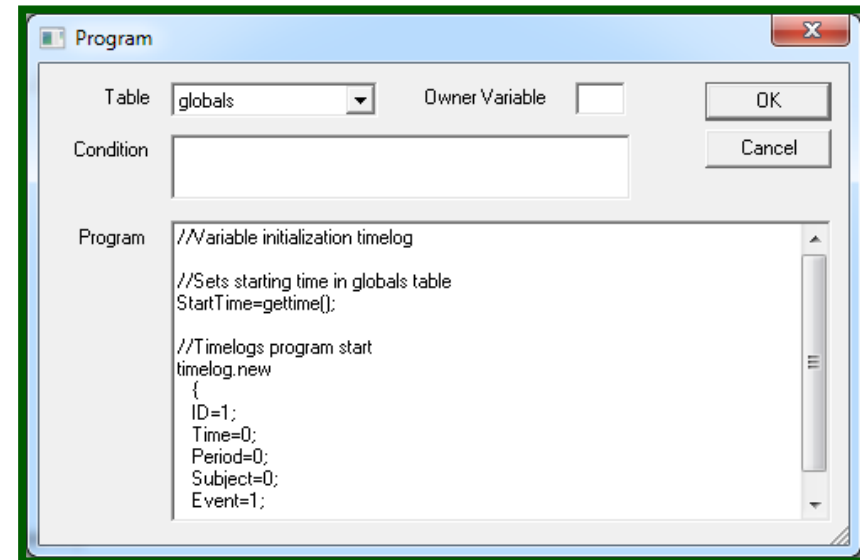
Timelog

- Create timelog table:
 - Allows custom logging of important events
- Record starting time and create first record:



The 'Table' dialog box is used to create a new table. It contains the following fields and options:

- Name:** A text field containing 'timelog'.
- Lifetime:** A group box with three radio buttons:
 - ☐ Period
 - ☐ Treatment
 - ☒ Session
- Program execution:** A group box with three radio buttons:
 - ☐ When first subject enters stage
 - ☒ Always when a subject enters stage
 - ☐ When last subject enters stage
- Used Variables (comma separated):** A text field containing 'Time,Period,Subject,Event'.
- Buttons:** 'OK' and 'Cancel' buttons are located on the right side.



The 'Program' dialog box is used to define a program. It contains the following fields and options:

- Table:** A dropdown menu showing 'globals'.
- Owner Variable:** An empty text field.
- Condition:** An empty text field.
- Program:** A text area containing the following code:

```
//Variable initialization timelog
//Sets starting time in globals table
StartTime=gettime();

//Timelogs program start
timelog.new
{
  ID=1;
  Time=0;
  Period=0;
  Subject=0;
  Event=1;
```
- Buttons:** 'OK' and 'Cancel' buttons are located on the right side.

- Create entries for all events (of interest):

Variable	Content
ID	Unique identifier of this entry
Time	Time in seconds since start of treatment
Period	Period number of the record
Subject	Subject event was triggered by, (0) if not applicable
Event	Event being logged: 1 Program start 2 Start stage Vote 3 Subject finished voting 4 Start stage VoteResult 5 Subject finished viewing VoteResult 6 Start stage Decision 7 Continue from screen Instructions(0) 8 Return to previous screen from Instructions(2) 9 Continue from screen Instructions(2) 10 Continue from screen ControlQuestion1 (3) 11 Return to previous screen from ControlQuestion1 (4) 12 Continue from screen ControlQuestion2 (4) 13 Return to previous screen from ControlQuestion2 (5) 14 Continue from screen ControlQuestion3 (5) 15 Return to previous screen from ControlQuestion3 (6) 16 Continue from screen ControlQuestion4 (6) 17 Return to previous screen from ControlQuestion4 (6) 18 Confirm decision in screen Decision (1) 19 Start stage DieRollStage 20 Start stage GeneralResults

Program

Table: globals Owner Variable: ☐

Condition:

Program

```
//Timelog
timelog.new
{
ID=timelog.maximum(ID)+1;
Time=gettime()-\StartTime;
Period=\Period;
Subject=0;
Event=6;
}
```

Program

Table: subjects Owner Variable: Subject

Condition:

Program

```
//Timelog
timelog.new
{
ID=timelog.maximum(ID)+1;
Time=gettime()-\StartTime;
Period=\Period;
Subject=Subject;
Event=9;
}
```

- Create entries for all events (of interest):

Variable	Content
ID	Unique identifier of this entry
Time	Time in seconds since start of treatment
Period	Period number of the record
Subject	Subject event was triggered by, (0) if not applicable
Event	Event being logged: 1 Program start 2 Start stage Vote 3 Subject finished voting

Period	ID	Time	Subject	Event
0	1	0	0	1
1	2	0.328	0	2
1	3	0.421	0	4
1	4	0.546	0	6
1	5	3.916	1	7
1	6	6.755	1	8
1	7	7.457	1	7
1	8	7.987	1	9
1	9	13.588	1	18
1	10	16.505	2	7
1	11	16.833	2	9
1	12	22.714	2	18
1	13	25.257	3	7
1	14	25.881	3	9
1	15	31.746	3	18
1	16	31.762	0	19
1	17	34.133	0	20

Program

Table: Owner Variable:

Condition:

Program

```
//Timelog
timelog.new
{
ID=timelog.maximum(ID)+1;
Time=gettime()-\StartTime;
Period=\Period;
Subject=0;
Event=6;
}
```

OK Cancel

Program

Table: Owner Variable:

Condition:

Program

```
//Timelog
timelog.new
{
ID=timelog.maximum(ID)+1;
Time=gettime()-\StartTime;
Period=\Period;
Subject=Subject;
Event=9;
}
```

OK Cancel

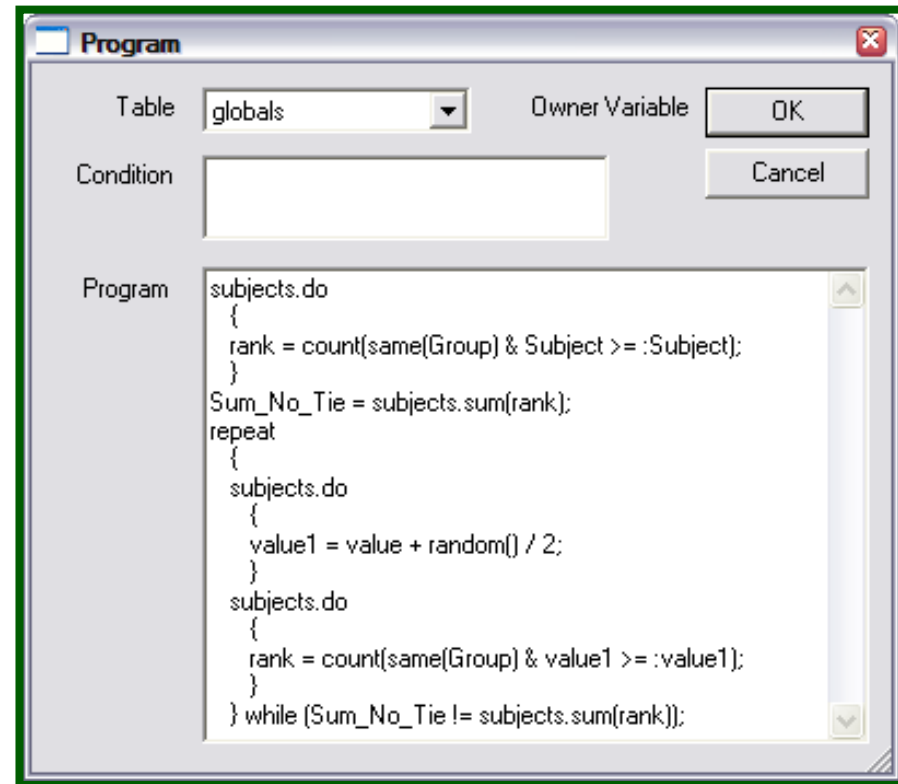
Experimenter subject*

- Implementing observer or experimenter subject (e.g. for die-throw)
- Variable that is 1 for observer and 0 for others
- Create specific observer stage displaying relevant variables or permitting input of a variable
- Use Participate to exclude observer from normal stages and exclude subjects from observer stage
- Use checkers to let experimenter allow subjects to continue step-by-step in training phase

* Source: z-Tree manual.

Calculating rank in z-Tree

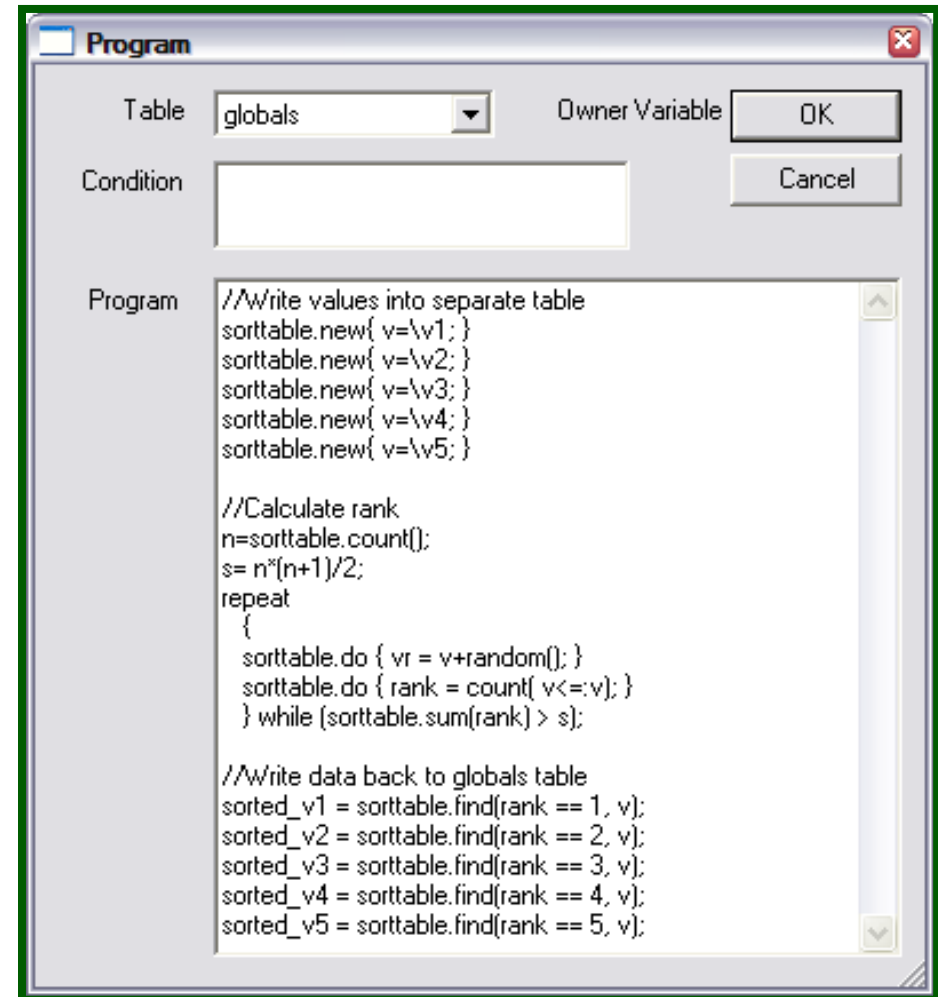
- Simple rank calculation (grouping is optional):
 - $\text{rank_low} = \text{count}(\text{same}(\text{Group}) \ \& \ \text{value} > : \text{value}) + 1;$
 - $\text{rank_high} = \text{count}(\text{same}(\text{Group}) \ \& \ \text{value} \geq : \text{value});$
- Rank calculation excluding ties (assuming value is integer):



Source: z-Tree Wiki.

Sorting values in z-Tree*

- Assume there are integers v1 to v5 in the globals table
- Strategy:
 - Put data into a table
 - Calculate the rank
 - Transfer the data back



* Source: z-Tree Wiki.

Calculation of payouts

- Set exchange rate in Background to 1
- Create variables in table globals.
 - exchangerate: The exchange rate in real CU/ECU
 - currency: A dummy variable for currency, e.g. 1 for EUR, 2 for USD...

**Return to
Basics of z-Tree Programming**

Item

Label

Variable: questionpayoff

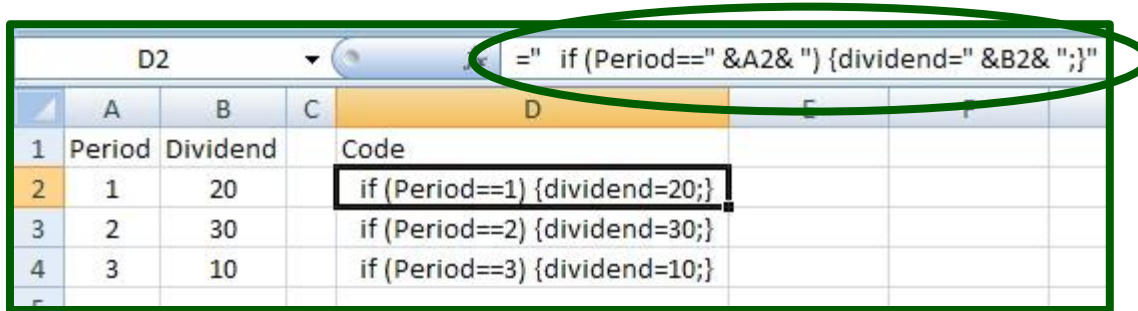
Layout: `<> !text: 0="This means that you will not receive additional payment for your quiz answers." 1="This means that you will receive an additional payment of <questionpayoff*\exchangerate | 1> <\currency|!text: 1="Euro" 2="US Dollars" 3="Danish Kroner" 4="Australian Dollars">."`

☐ Input

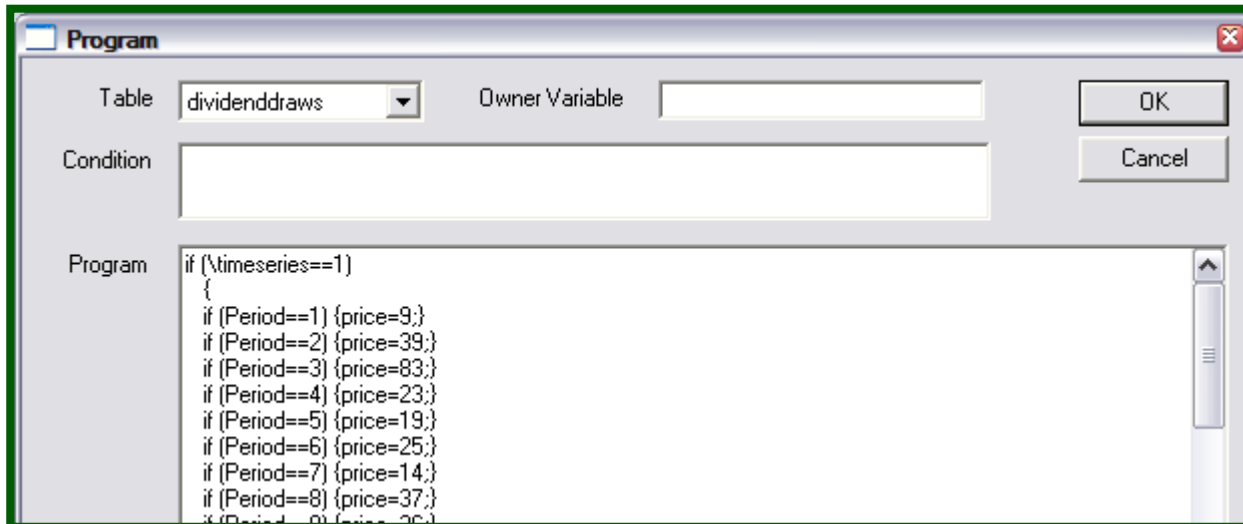
Only one place for changing these parameters (in the program in the `globals` table)!

Generating repetitive code with Excel

- Use MS Excel to generate repetitive code:



	A	B	C	D
1	Period	Dividend	Code	
2	1	20	if (Period==1) {dividend=20;}	
3	2	30	if (Period==2) {dividend=30;}	
4	3	10	if (Period==3) {dividend=10;}	



Program

Table: dividenddraws

Owner Variable:

Condition:

Program:

```
if (\timeseries==1)
{
  if (Period==1) {price=9;}
  if (Period==2) {price=39;}
  if (Period==3) {price=83;}
  if (Period==4) {price=23;}
  if (Period==5) {price=19;}
  if (Period==6) {price=25;}
  if (Period==7) {price=14;}
  if (Period==8) {price=37;}
  if (Period==9) {price=26;}
}
```

Generating repetitive code with Excel

- Complex code generation

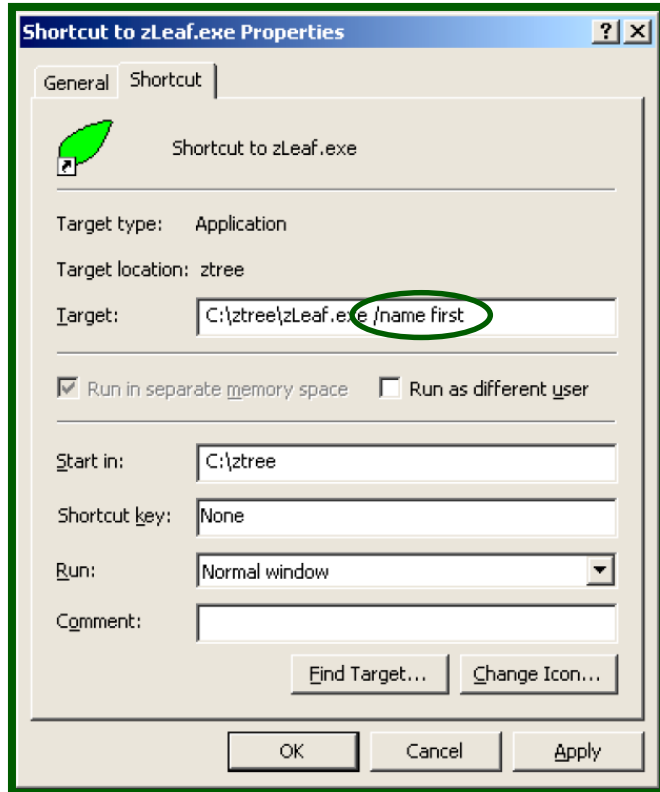
Return to
Import and Export

C2		fx	= "later (round("&A2&"/daylengthbase*daylength,1)) do"
A	B	C	
1	Time	Price	Code
2	40	365,29	later (round(40/daylengthbase*daylength,1)) do
3			{
4			signals.new
5			{
6			signal=365;
7			signaltime=gettime()-summary.find(same(Period),timeperiodstarted);
8			hour=\daybeginhour+rounddown(\dayhours*(signaltime/\daylength),1);
9			minute=rounddown((signaltime-(hour-\daybeginhour)/\dayhours*\daylength)/(\day
10			weekday=1;
11			}
12			}
13	72	368,33	later (round(72/daylengthbase*daylength,
14			{
15			signals.new
16			{
17			signal=368;
18			signaltime=gettime()-summary.find(same(Period),timeperiodstarted);
19			hour=\daybeginhour+rounddown(\dayhours*(signaltime/\daylength),1);
20			minute=rounddown((signaltime-(hour-\daybeginhour)/\dayhours*\daylength)/(\day
21			weekday=1;
22			}
23			}
24	85	364,90	later (round(85/daylengthbase*daylength,1)) do
25			{
26			signals.new
27			{
28			signal=365;
29			signaltime=gettime()-summary.find(same(Period),timeperiodstarted);
30			hour=\daybeginhour+rounddown(\dayhours*(signaltime/\daylength),1);
31			minute=rounddown((signaltime-(hour-\daybeginhour)/\dayhours*\daylength)/(\day

```
zTree - [#daynight_baseline_overnight.ztt]
File Edit Treatment Run Tools View ?
+ StartingScreen != (-1)N
+ TradeScreenWE != (wlength)A
+ TradeScreenNight != (nightlength)A
+ TradeScreenDay != (daylength)A
- summary.do {timeperiodstartedprevious=timeperiodstarted; ...}
+ globals(Period==1).do { //Generates Signals Period 01 ... }
+ globals(Period==2).do { //Generates Signals Period 02 ... }
+ globals(Period==3).do { //Generates Signals Period 03 ... }
+ globals(Period==4).do { //Generates Signals Period 04 ... }
+ globals(Period==5).do { ... }
  //Generates Signals Period 05
  summary.do(dividende=1;)
  //Writes new signals into table signals at the time specified in the later function
  later (round(31/daylengthbase*daylength,1)) do
  {
    signals.new
    {
      signal=384;
      signaltime=gettime()-summary.find(same(Period),timeperiodstarted);
      hour=\daybeginhour+rounddown(\dayhours*(signaltime/\daylength),1);
      minute=rounddown((signaltime-(hour-\daybeginhour)/\dayhours*\daylength)/(\daylength/(\dayhours*60)),1);
      weekday=5;
    }
  }
  later (round(48/daylengthbase*daylength,1)) do
  {
    signals.new
    {
      signal=393;
      signaltime=gettime()-summary.find(same(Period),timeperiodstarted);
      hour=\daybeginhour+rounddown(\dayhours*(signaltime/\daylength),1);
      minute=rounddown((signaltime-(hour-\daybeginhour)/\dayhours*\daylength)/(\daylength/(\dayhours*60)),1);
      weekday=5;
    }
  }
  later (round(55/daylengthbase*daylength,1)) do
  {
    signals.new
```

Starting multiple z-leafs for testing

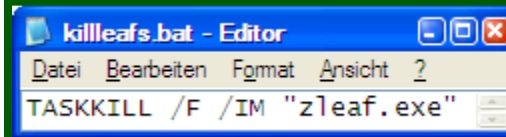
- Either create shortcuts:



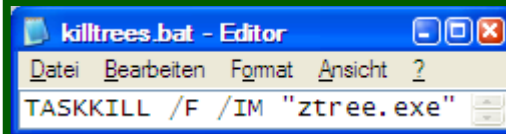
- ...or use the batch files on the following slide

Shutting down z-leafs and z-trees

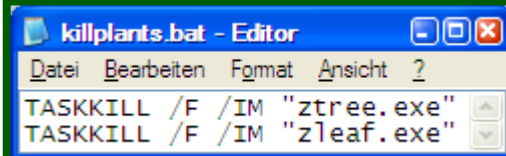
- killleafs.bat shuts down all running z-leafs:
- killtrees.bat shuts down all running z-trees (process often remains active after closing z-tree):
- killplants.bat shuts down both:
- Note that growleafs.bat automatically shuts down any remaining z-leafs before starting them anew:



```
killleafs.bat - Editor
Datei Bearbeiten Format Ansicht ?
TASKKILL /F /IM "zleaf.exe"
```

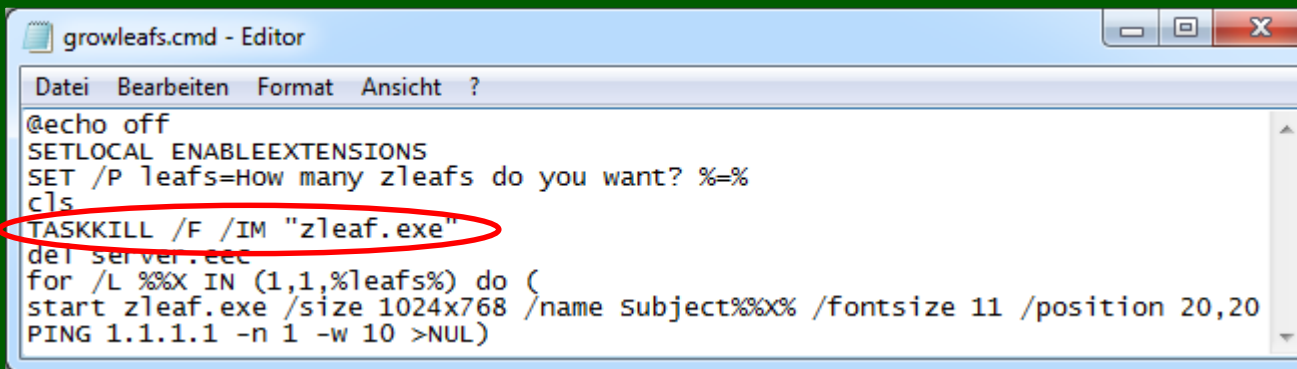


```
killtrees.bat - Editor
Datei Bearbeiten Format Ansicht ?
TASKKILL /F /IM "ztree.exe"
```



```
killplants.bat - Editor
Datei Bearbeiten Format Ansicht ?
TASKKILL /F /IM "ztree.exe"
TASKKILL /F /IM "zleaf.exe"
```

Return to
Public Goods Game Tutorial



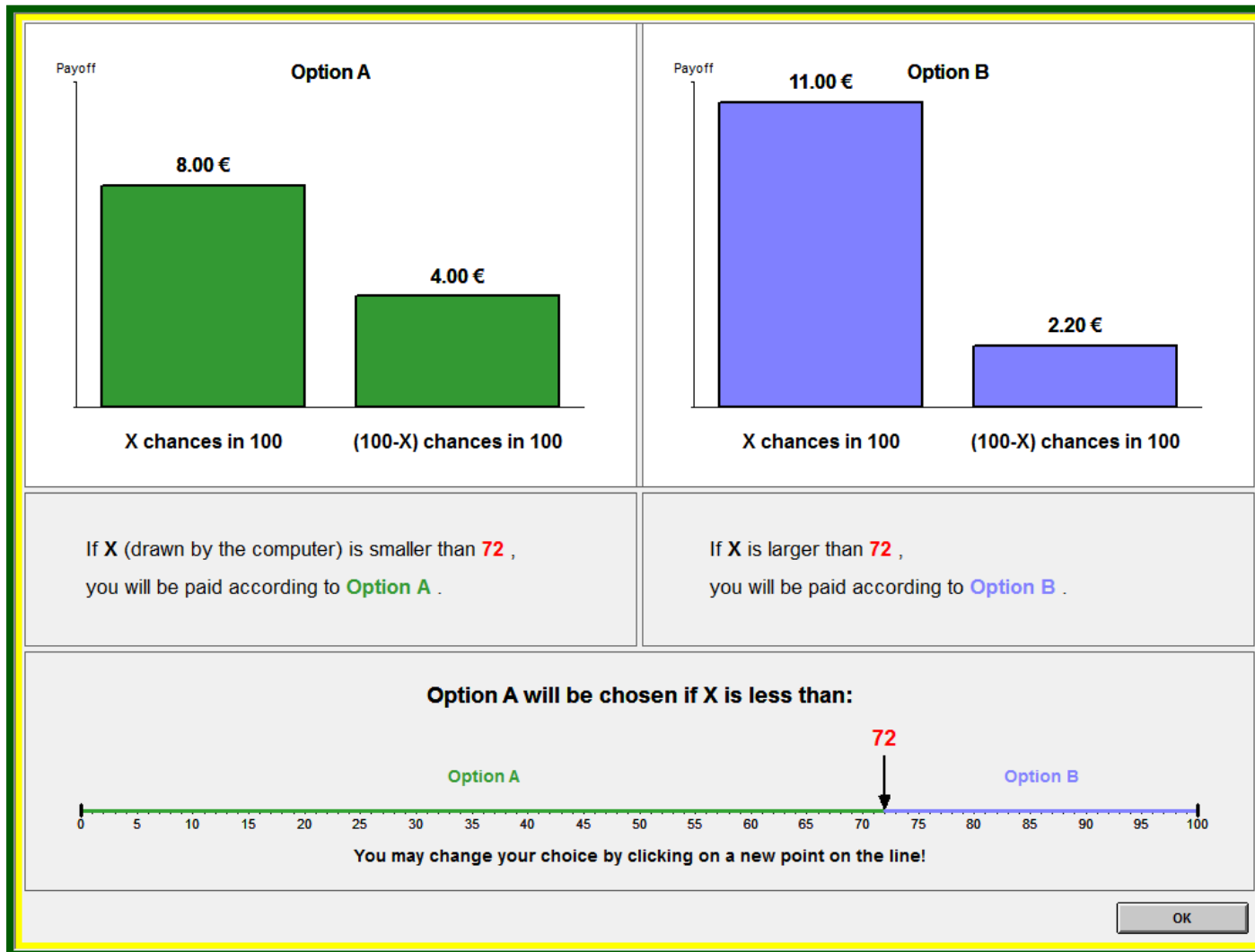
```
growleafs.cmd - Editor
Datei Bearbeiten Format Ansicht ?
@echo off
SETLOCAL ENABLEEXTENSIONS
SET /P leafs=How many zleafs do you want? %=%
cls
TASKKILL /F /IM "zleaf.exe"
del server.eec
for /L %%X IN (1,1,%leafs%) do (
start zleaf.exe /size 1024x768 /name Subject%%X% /fontsize 11 /position 20,20
PING 1.1.1.1 -n 1 -w 10 >NUL)
```

Graphics Teaser

- Capabilities
- Plot items
- Demo treatments

Graphics teaser

- z-Tree offers graphics capabilities since version 3.0

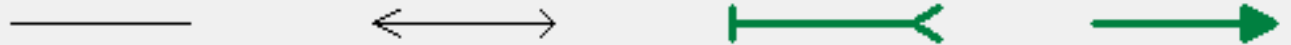


Selection of plot items

8 points



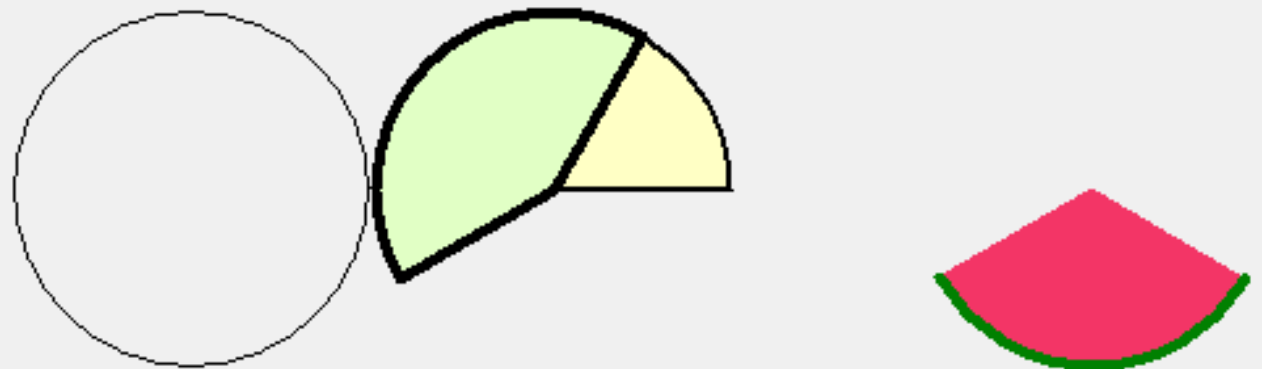
4 lines



3 rects



4 pies



Further information

- Demo treatments demonstrate graphics capabilities:
 - `colortriangle.ztt`
 - `movepointdemo.ztt`
 - `selectlinkdemo.ztt`
 - `z-draw.ztt`
 - `animatetreedemo.ztt`
- More information on graphics in z-Tree:
 - z-Tree lecture by Maria Bigoni:
http://www2.dse.unibo.it/bigoni/courses/ztree/Lecture_notes_handout.pdf
 - z-Tree Wiki:
<https://www.uzh.ch/iew/ztree/ssl-dir/wiki/>

Single-Unit CDA with Limit and Market Orders

- Concept
 - Interface layout
 - Contracts table record
- Parameter initialization
- Limit orders
- Market orders
- Limiting order legality

Concept: Interface layout

- Goal: Continuous market for buyers and sellers of a uniform good
- Every trader can buy and sell
- Every order is for one unit

Cash: 1000 Shares: 100

Price
Cancel Sell

Price	Volume
Buy Cancel	

Price

Submit sell order

Submit buy order

Markets and auctions: contracts table*

- Flexible number of records
- Adding records
 - Contract creation boxes
 - Command: `contracts.new { x=1; }`
- Changing records
 - In contract list boxes and contract grid boxes
 - Command: `contracts.do { x=2; }`
- Deleting records
 - Marking a contract as deleted (e.g. `status = 3;`)

* Adapted from lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

- Buttons in contract creation box and contract list box
 - Can contain checkers (e.g. $\text{price} * \text{volume} < \text{cash}$)
 - Operate only on new/selected record
 - Access to subject who pressed the button using scope operator
 - As if in `contracts.new/do {...}`
 - E.g. set subject using: `CreatorOrSelector = :Subject;`

* Adapted from lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

Concept: Contracts table record

- Example record
 - ID = 1;
 - price = 23;
 - volume = 1;
 - status = 0;
 - type = 1;
 - buyer = 1;
 - seller = 0;
 - offertime = 2.381;
 - statustime = 18.221;
- Variable status
 - 0 ... open offer
 - 1 ... traded
 - 2 ... cancelled
 - 3 ... invalidated
 - 4 ... expired

Concept: Contracts table record

- Variable type
 - +1 ... buy offer
 - -1 ... sell offer
- Open buy offer characterized by:
 $\text{type} == 1 \ \& \ \text{status} == 0$
- Open sell offer characterized by:
 $\text{type} == -1 \ \& \ \text{status} == 0$
- Transaction characterized by:
 $\text{status} == 1$

Concept: Contracts table record

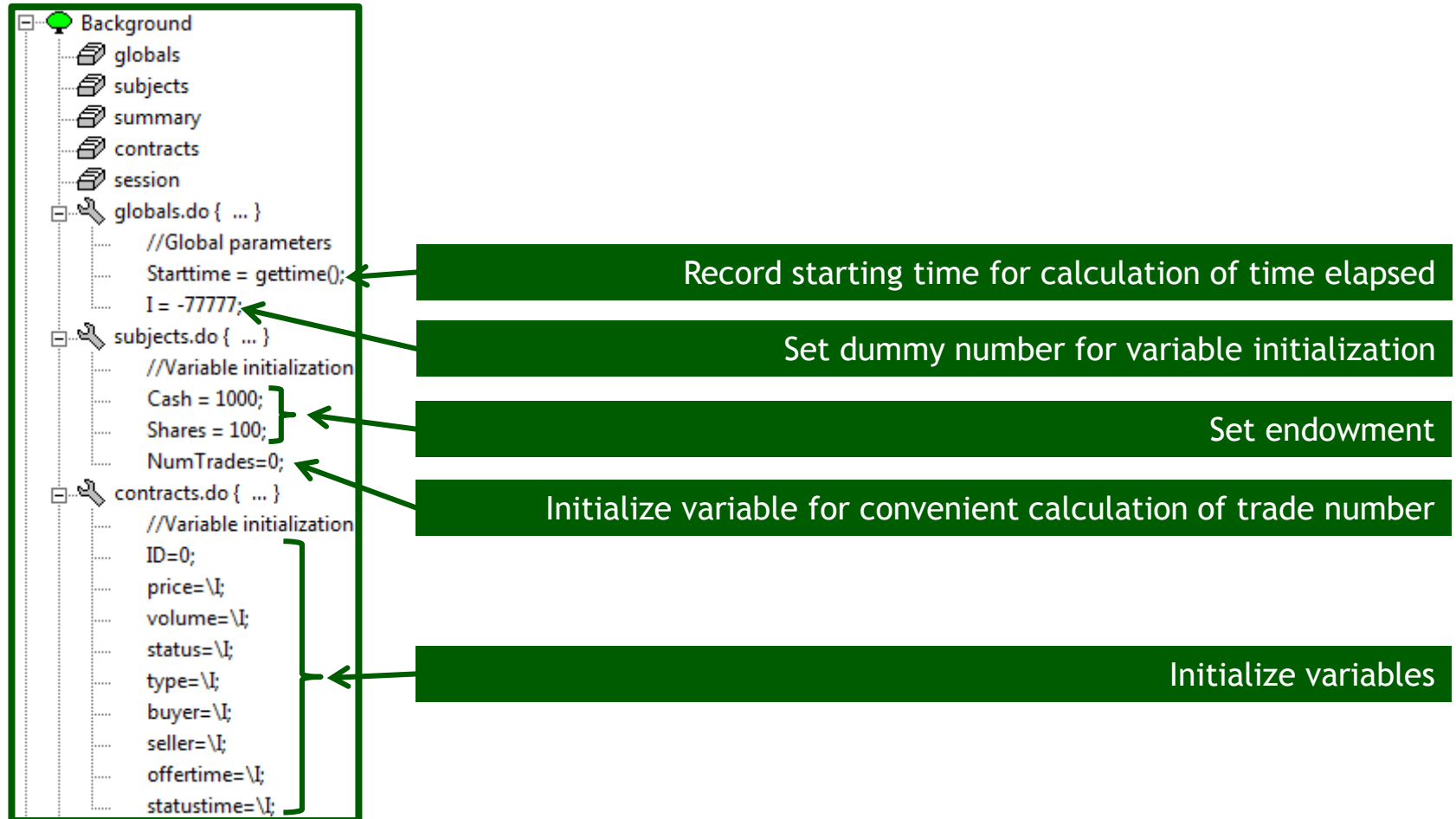
- Example records:

ID	price	volume	status	type	buyer	seller	offertime	statustime
1	23	1	0	-1	0	3	5.304	-77777
2	15	1	1	1	2	1	7.221	18.475
3	21	1	3	-1	0	1	12.132	18.475
4	30	1	2	1	5	0	15.100	16.224

- Record 1: Outstanding sales offer from subject 3
- Record 2: Subject 1 sold after a buy offer from subject 2
- Record 3: Invalidated sell offer from subject 1
- Record 4: Cancelled buy offer from subject 5

Parameter initialization

- Variable initialization:



Limit order input

Contract Creation Box

Name

Contract maker

☒ With frame

Width [p/%]

235p

Height [p/%]

150p

Distance to the margin [p/%]

0p

Adjustment to the remaining box

☐ left

☐ top

☐ right

☒ bottom

OK

Cancel

Display condition

Table

contracts

Num. records

1

Records' display

☐ Labels on the top

☒ Labels on the left

☐ Empty records allowed

☐ All records empty allowed

Buttons

Position

☐

☐

☐

☐

☐

☐

☐

☐

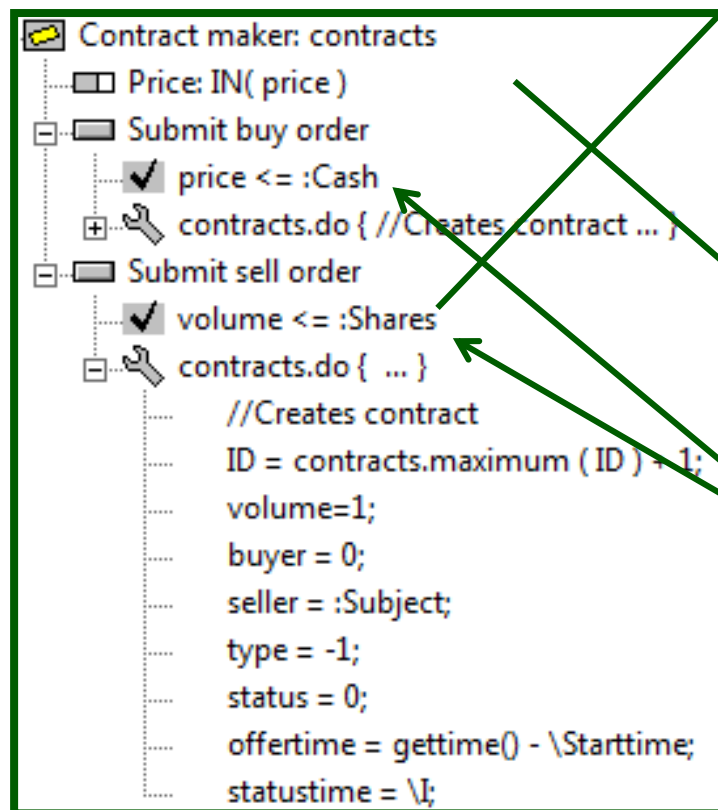
☒

Arrangement

☒ In rows

☐ In columns

Limit order input



Checker

Condition: volume <= :Shares

Message (This message appears if the condition is not satisfied.): You do not have enough shares to submit this order!

"yes"-Button:

"no"-Button: OK

If there is only a "yes"-button, then the message appears and after pressing this button the input is accepted.

If there is only a "no"-button, then the message appears and after pressing this button the input is rejected.

If there is a "yes" AND a "no"-button, then the message should contain a question. Pressing the YES button accepts the input pressing NO rejects it.

Submit sell order

Submit buy order

Order display

Contract Box [X]

Name ☒ With frame

Width [p/%]

Height [p/%]

Distance to the margin [p/%]

Adjustment to the remaining box

☐ left ☐ top ☐ right
☐ bottom

OK
Cancel

Display condition

Table

Owner var.

Condition

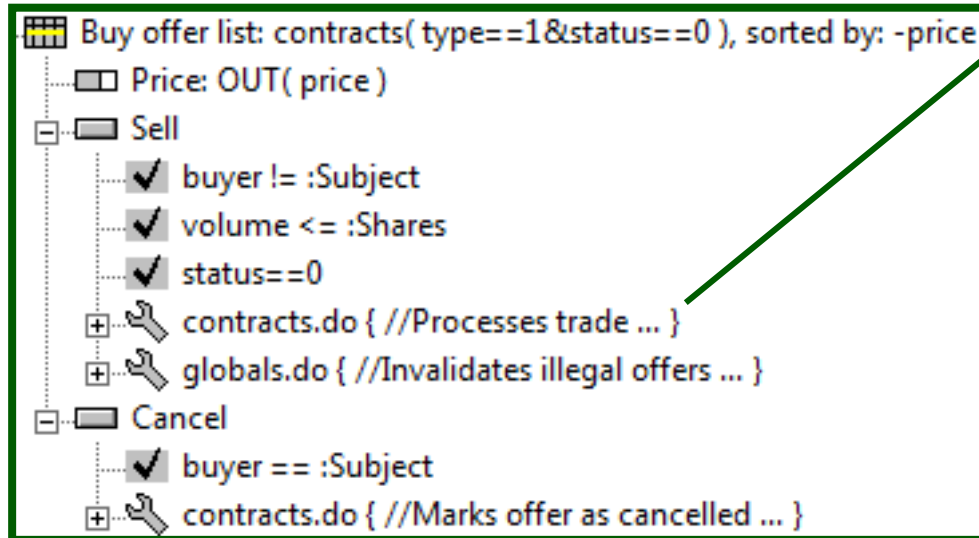
Sorting

Scrolling ☐ To beginning ☒ To end

☒ Mark best foreign contract

Buttons

Market order submission



```
//Processes trade

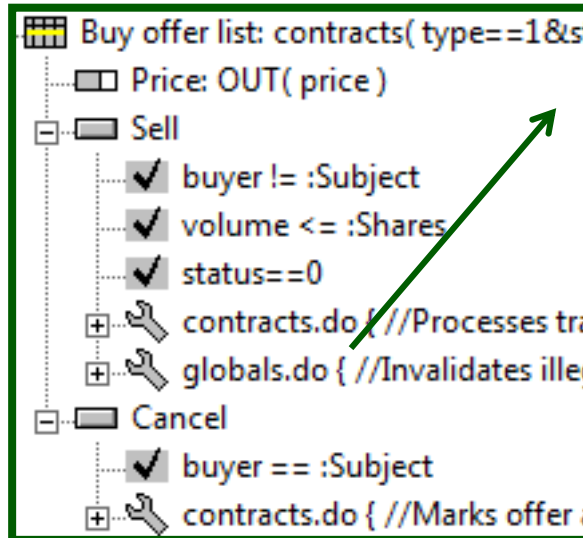
//Updates contract information
seller = :Subject;
status = 1;
statustime = gettime() - \Starttime;

//Updates cash and share holdings
subjects.do {

    //Updates seller
    if ( Subject == :seller ) {
        NumTrades = NumTrades + 1;
        Cash = Cash + :price * :volume;
        Shares = Shares - :volume;
    }

    //Updates buyer
    if ( Subject == :buyer ) {
        NumTrades = NumTrades + 1;
        Cash = Cash - :price * :volume;
        Shares = Shares + :volume;
    }
}
```

Market order submission



```
//Invalidates illegal offers

//Loops over all contracts
contracts.do {

    //Looks only at outstanding offers
    if ( status == 0 ) {

        //Only buy offers
        if ( type == 1 ) {

            //Invalidates buy offers if proposer has insufficient money
            if ( price * volume > subjects.find ( Subject == :buyer , Cash ) ) {
                status = 3;
                statustime = gettime() - \Starttime;
            }
        }

        //Only sell offers
        if ( type == -1 ) {
            //Invalidates sell offers if proposer has insufficient shares
            if ( volume > subjects.find ( Subject == :seller , Shares ) ) {
                status = 3;
                statustime = gettime() - \Starttime;
            }
        }
    }
}
}
```

Market order submission



```
//Marks offer as cancelled  
status = 2;  
statustime = gettime() - \Starttime;
```

Order display

 Buy offer list: `contracts( type==1&status==0 )`, sorted by: `-price`

- ☐ Price: OUT(price)
- ☒ Sell
 - ☒ `buyer != :Subject`
 - ☒ `volume <= :Shares`
 - ☒ `status==0`
 - ☐  `contracts.do { //Processes trade ... }`
 - ☐  `globals.do { //Invalidates illegal offers ... }`
- ☒ Cancel
 - ☒ `buyer == :Subject`
 - ☐  `contracts.do { //Marks offer as cancelled ... }`

Price	Volume
30	13
25	3
23	5
Cancel Sell	

Price	Volume
34	7
35	2
38	9
42	1
Buy Cancel	

Link to
Transacting best outstanding order
in
Order Legality

Extension to Multi-Unit CDA

- Concept
- Interface design
- Partial order invalidation

Exercise - Market with multi-unit trading

- Write down how you would structure the tables and variables of a market to fulfill the following criteria:
- Subjects can offer to buy and sell multiple units of a uniform good, entering a price and volume
- Partial execution is possible
- No information is lost:
 - Who made the first offer, who accepted
 - Time where the offer and acceptance were made
 - Initial volume, how much was exchanged per transaction

- Create record in table offers for every order
- Upon new order entry, check transactability
- Create new record in table transactions for every transaction based on new order
- Variable offerID in transactions contains ID of corresponding offers-table record
- Variable acceptanceID in transactions contains ID of order which led to transaction with previously submitted order offerID

Offers and transaction table records

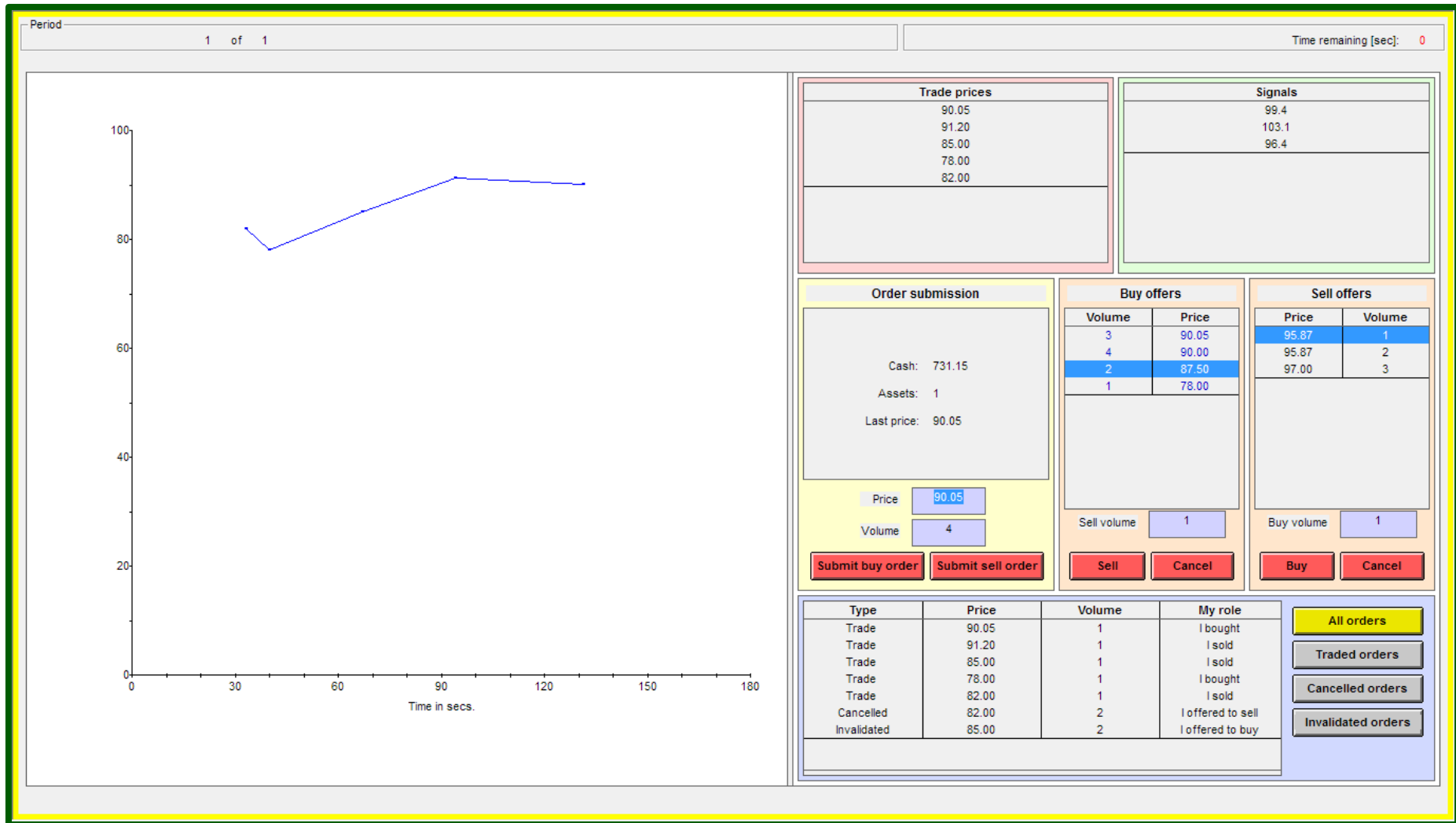
- Table offers

- ID
- price
- volume
- type (-1, 1)
- status (0, 1, 2, 3, 4)
- execution (0, volume)
- offerer
- offertime
- statustime

- Table transactions

- offerID (offers.ID)
- acceptanceID
- accepterID
- volume
- time
- (price)

Interface design



Source: GIMS.

Order Legality

- Transacting best outstanding order
- No-crossing rule
- Bid-ask-improvement rule
- Partial order invalidation

Transacting best outstanding order

- Write down the code for a rule to ensure that the subject buys (sells) for the lowest (highest) price when accepting an outstanding offer.

Checker	
Condition	<code>price == contracts.minimum (type == -1 & status == 0 & seller != :Subject, price)</code>
Message (This message appears if the condition is not satisfied.)	You can buy for a lower price!
"yes"-Button	
"no"-Button	OK

If there is only a "yes" button, then the message appears

Checker	
Condition	<code>price == contracts.maximum (type == 1 & status == 0 & buyer != :Subject, price)</code>
Message (This message appears if the condition is not satisfied.)	You can sell for a higher price!
"yes"-Button	
"no"-Button	OK

If there is only a "yes" button, then the message appears

No-crossing rule

- Write down the condition for a checker which ensures that the price of a new buy (sell) offer is lower (higher) than the lowest existing sell (highest buy) offer price.

Checker	
Condition	<code>price < contracts.minimum (type == -1 & status == 0, price)</code>
Message (This message appears if the condition is not satisfied.)	You cannot offer to buy for a higher price than the lowest sell offer price. Click "Buy" instead!
"yes"-Button	
"no"-Button	OK
If there is only a "yes"-button, then the message appears	

Checker	
Condition	<code>price > contracts.maximum (type == 1 & status == 0, price)</code>
Message (This message appears if the condition is not satisfied.)	You cannot offer to sell for a lower price than the highest buy offer price. Click "Sell" instead!
"yes"-Button	
"no"-Button	OK
If there is only a "yes"-button, then the message appears	

Bid-ask improvement rule*

- Write down the code for a bid-ask improvement rule (i.e. a new bid must be higher than the current highest bid price and a new ask price must be lower than the current lowest ask price.)

Return to
Single-Unit CDA

Checker	
Condition	<code>price > contracts.maximum (type == 1 & status == 0, price)</code>
Message (This message appears if the condition is not satisfied.)	A new buy offer price must be greater than the highest existing buy offer price!
"yes"-Button	
"no"-Button	OK

Condition	<code>price < contracts.minimum (type == -1 & status == 0, price)</code>
Message (This message appears if the condition is not satisfied.)	A new sell offer price must be smaller than the lowest existing sell offer price!
"yes"-Button	
"no"-Button	OK

If there is only a "yes" button, then the message appears.

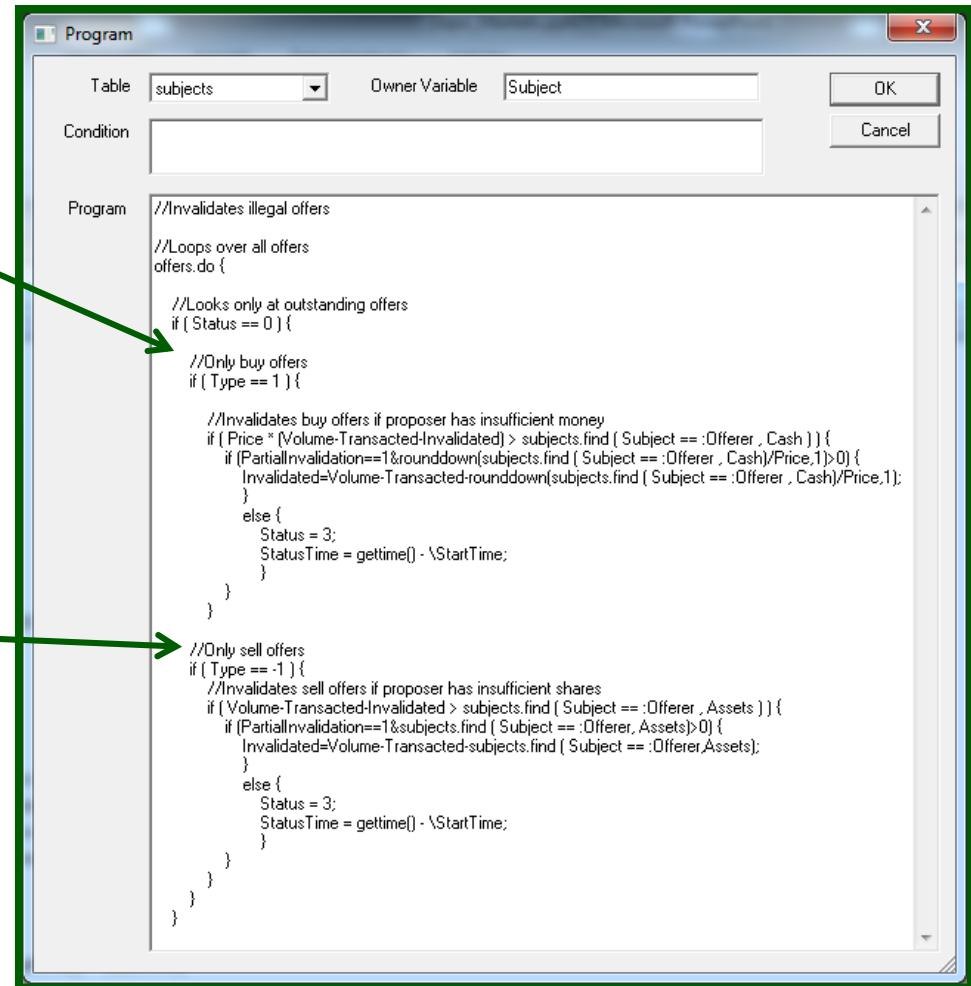
* Inspired by lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

Partial invalidation of pre-existing orders

- Invalidate only invalid part of total order

- Reduce number of assets to be bought if cash is insufficient

- Reduce number of asset to be sold if assets are insufficient



```
Program
Table: subjects Owner Variable: Subject
Condition:
Program
//Invalidates illegal offers
//Loops over all offers
offers.do {
  //Looks only at outstanding offers
  if (Status == 0) {
    //Only buy offers
    if (Type == 1) {
      //Invalidates buy offers if proposer has insufficient money
      if (Price * (Volume-Transacted-Invalidated) > subjects.find ( Subject == :Offerer , Cash )) {
        if (PartialInvalidation==1&rounddown(subjects.find ( Subject == :Offerer , Cash)/Price,1)>0) {
          Invalidated=Volume-Transacted-rounddown(subjects.find ( Subject == :Offerer , Cash)/Price,1);
        }
        else {
          Status = 3;
          StatusTime = getTime() - \StartTime;
        }
      }
    }
  }
  //Only sell offers
  if (Type == -1) {
    //Invalidates sell offers if proposer has insufficient shares
    if (Volume-Transacted-Invalidated > subjects.find ( Subject == :Offerer , Assets )) {
      if (PartialInvalidation==1&subjects.find ( Subject == :Offerer , Assets)>0) {
        Invalidated=Volume-Transacted-subjects.find ( Subject == :Offerer , Assets);
      }
      else {
        Status = 3;
        StatusTime = getTime() - \StartTime;
      }
    }
  }
}
```

Partial invalidation of pre-existing orders

```
//Invalidates illegal offers

//Loops over all offers
offers.do {

    //Looks only at outstanding offers
    if ( Status == 0 ) {

        //Only buy offers
        if ( Type == 1 ) {

            //Invalidates buy offers if proposer has insufficient money
            if ( Price * (Volume-Transacted-Invalidated) > subjects.find ( Subject == :Offerer , Cash ) ) {
                if (PartialInvalidation==1&rounddown(subjects.find ( Subject == :Offerer , Cash)/Price,1)>0) {
                    Invalidated=Volume-Transacted-rounddown(subjects.find ( Subject == :Offerer , Cash)/Price,1);
                }
                else {
                    Status = 3;
                    StatusTime = gettime() - \StartTime;
                }
            }
        }
    }
}
```

Partial invalidation of pre-existing orders

```
//Only sell offers
if ( Type == -1 ) {
    //Invalidates sell offers if proposer has insufficient shares
    if ( Volume-Transacted-Invalidated > subjects.find ( Subject == :Offerer , Assets ) ) {
        if ( PartialInvalidation==1&subjects.find ( Subject == :Offerer , Assets)>0 ) {
            Invalidated=Volume-Transacted-subjects.find ( Subject == :Offerer,Assets);
        }
        else {
            Status = 3;
            StatusTime = gettime() - \StartTime;
        }
    }
}
}
```

Order Matching and Execution Time

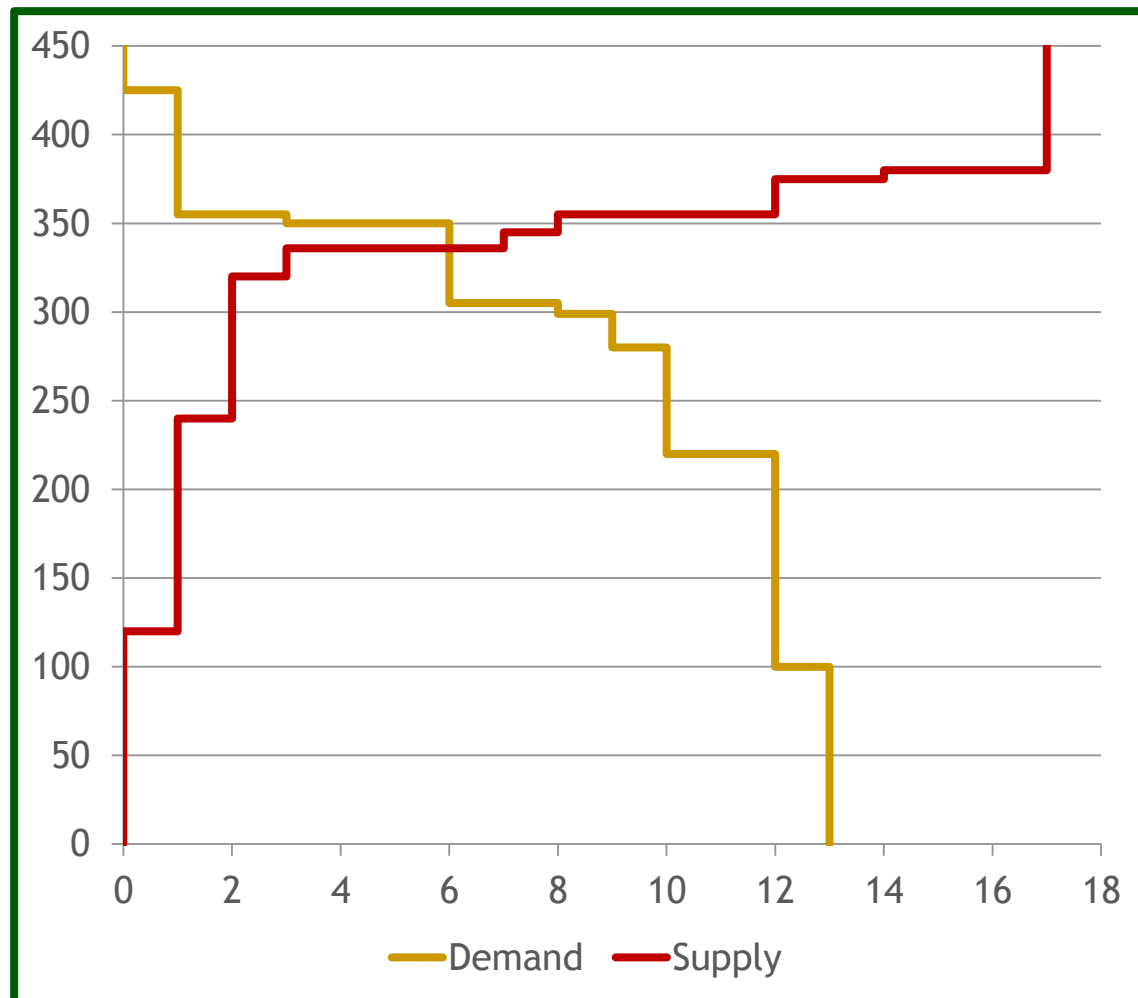
- Effective programming
- Call auction order matching

- Tradeoff: Memory use vs. execution speed
 - Always calculate and use expressions , e.g. `subjects.count()`, vs.
 - Calculate once and save in variable, e.g. `NumSubjects=subjects.count()`
- Tradeoff: Code length vs. comprehensibility
 - Re-use existing variables, e.g. `Buyer/Seller== -1, -2,...` vs.
 - Create new variables, e.g. `Status==0, 1,...`

Call auction price determination

- Call auction price determination:

Bids		Asks	
Vol	Price	Vol	Price
1	425	1	120
2	355	1	240
2	350	1	320
1	350	4	336
2	305	1	345
1	299	4	355
1	280	2	375
2	220	3	380
1	100	1	650



Order matching background

- Limit vs. market orders
- Priority:
 - Bids (asks) with price greater (less) than settlement price
 - Highest bid (lowest ask) price
 - Earliest submission time
 - Market order
 - Earliest submission time
 - Bid (ask) with price==settlement price
 - Earliest submission time
 - Random choice

Order matching procedure

- Steps for order matching:
 - Find open ask with highest priority
 - Find open bid with highest priority
 - Write contract into contracts table, with $\min(\text{Volume})$
 - Mark orders as (partially) transacted
 - Update buyer's/seller's cash and asset balances
 - Repeat

Order matching procedure

- Solution 1: Use no extra tables (only offers and contracts)
 - See GraphSettlementExtensive.graphml
 - Everything needs to be done in a single program
 - Always needs to loop through all records in offers table
 - Very inefficient

Order matching procedure

- Solution 2: Use interim tables (tbsa, tbsb, settlement)
 - Writes asks into tbsa in order of priority
 - Writes bids into tbsb in order of priority
 - Match tbsa with tbsb to create settlement
 - Do bookkeeping based on settlement
 - See GraphSettlementOptimized.graphml
 - Matching makes use of sorted tables tbsa, tbsb
 - Bookkeeping runs through settlement once
 - Easy to track program progress and errors

Order matching procedure

- Solution 2: Use interim tables (tbsa, tbsb, settlement)

```
CallAuctionResults = | = (if(\Period==\NumPeriods&(\CRTAfter==1|FinLit==1),-1,\TimeCallAuctionResults))
+ subjects.do { Participate=if(\ForcedExitExperiment+\ForcedExitPeriod>0,0,1)*(1-\PracticePeriod*(1-\Testmode1))*\CallAuction; }
+ globals(\CallAuction==1).do { //Timelog ... }
+ globals(\CallAuction==1).do { //Fills pricedetermination table ... }
+ globals(\CallAuction==1).do { //Calculates aggregate transaction variables ... }
+ globals(\CallAuction==1).do { //Determines price ... }
+ globals(\CallAuction==1).do { //Timelog ... }
+ globals(\CallAuction==1&\PriceDeterminationRule!=0).do { //Settlement 1: Filling tbs tables with BTE limit orders ... }
+ globals(\CallAuction==1&\PriceDeterminationRule!=0).do { //Settlement 2: Filling tbs tables with market orders if \MarketOrderPriority==1 ... }
+ globals(\CallAuction==1&\PriceDeterminationRule!=0).do { //Settlement 3: Filling tbs tables with AE limit orders ... }
+ globals(\CallAuction==1&\PriceDeterminationRule!=0).do { //Settlement 4: Filling tbs tables with market orders if \MarketOrderPriority==0 ... }
+ globals(\CallAuction==1&\PriceDeterminationRule!=0).do { //Settlement 5: Matches orders and fills settlement table ... }
+ globals(\CallAuction==1&\PriceDeterminationRule!=0).do { //Settlement 6: Updates all tables using settlement table data ... }
+ globals(\CallAuction==1).do { //Timelog ... }
+ globals(\CallAuction==1).do { //Dividend draw ... }
+ subjects(IsExperimenter==0&\CallAuction==1).do { //Updates subjects' cash and writes profit ... }
+ summary(\CallAuction==1).do { //Writes statistics into summary table ... }
+ Active screen
+ Waiting screen
```

Exercises

- A simple guessing game
- Ultimatum game
- Trust game
- Keynes' beauty contest
- Dutch auction
- Random ending period
- Progression within a period

A simple guessing game*

- Program the following treatment:
- The computer draws a random integer between 1 and 10 (y)
- Each subject enters an integer between 1 and 10 (x)
- If x equals y , the subject receives 100 ECU
- Display the outcome of the treatment

- Hints:
 - $y = \text{roundup}(\text{random()} * 10, 1);$
 - $\text{Profit} = \text{if}(x == y, 100, 0);$

* Adapted from lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

Ultimatum Game 1/2

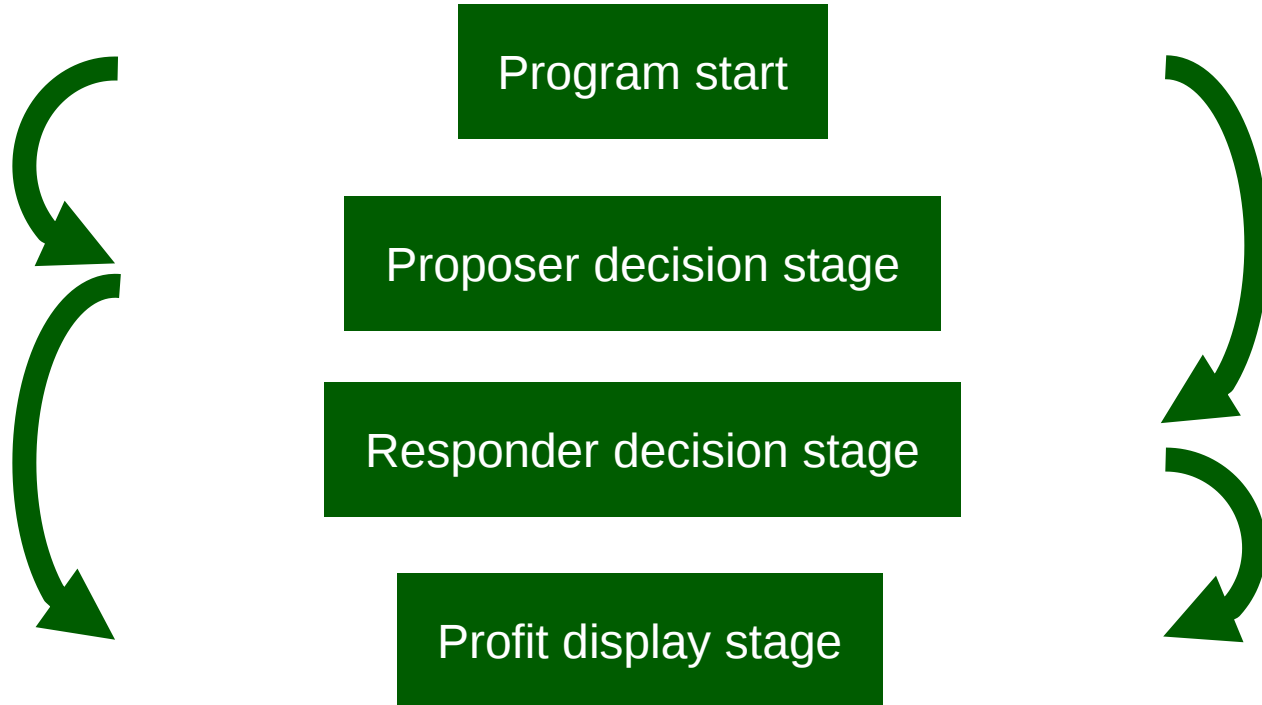
- Program the following treatment:
- 2 subjects, 1 period
- Subject A receives 10 euros and decides how much to transfer to subject B
- Subject B can either accept or reject the offer
 - If B accepts, A gets what he kept and B gets what he was given
 - If B rejects, both players receive 0
- Display the outcome of the treatment

Ultimatum Game 2/2

- Subject A

Structure

Subject B



- Hint: $\text{Participate} = 1 - \text{Proposer}$;

Trust Game

- Program the following treatment:
- 2 subjects, 1 period
- Subject A receives 10 euros and decides how much to transfer to subject B
- The amount transferred is multiplied by 3 when it arrives in subject B's account
- Subject B decides how much to transfer back to subject A
- Display the outcome of the treatment

Keynes' beauty contest 1/2

- Program the following treatment:
- 9 subjects participate in an experiment with 3 periods
- Subjects are randomly matched and rematched into groups of 3 (absolute stranger matching)
- Each subject enters an integer x between 0 and 100
- The subject whose x is closest to 0.5 times the group average of x receives 100 ECU
- Display the outcome of the treatment

Keynes' beauty contest 2/2

- Hints:
- `distance=abs(x-0.5*subjects.average(same(Group),x));`
- `Winner=if(distance==subjects.minimum(same(Group), /*
*/ distance),1,0);`
- Extra tasks:
- In case there would be more than one winner, make sure that one of them is randomly chosen to receive the 100 ECU
- Implement an experimenter subject which sees the entries made by the other subjects (contract list)

Dutch auction 1/2*

- Program the following treatment:
- There are 15 subjects, in 3 groups of 5 subjects
- Each subject is endowed with 100 ECU
- Each subject can buy a maximum of 1 unit of a good
- The good has different values for different subjects
 - A subject's valuation is randomly determined between 10 and 90
 - Each subject is informed only of their own valuation

* Adapted from lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

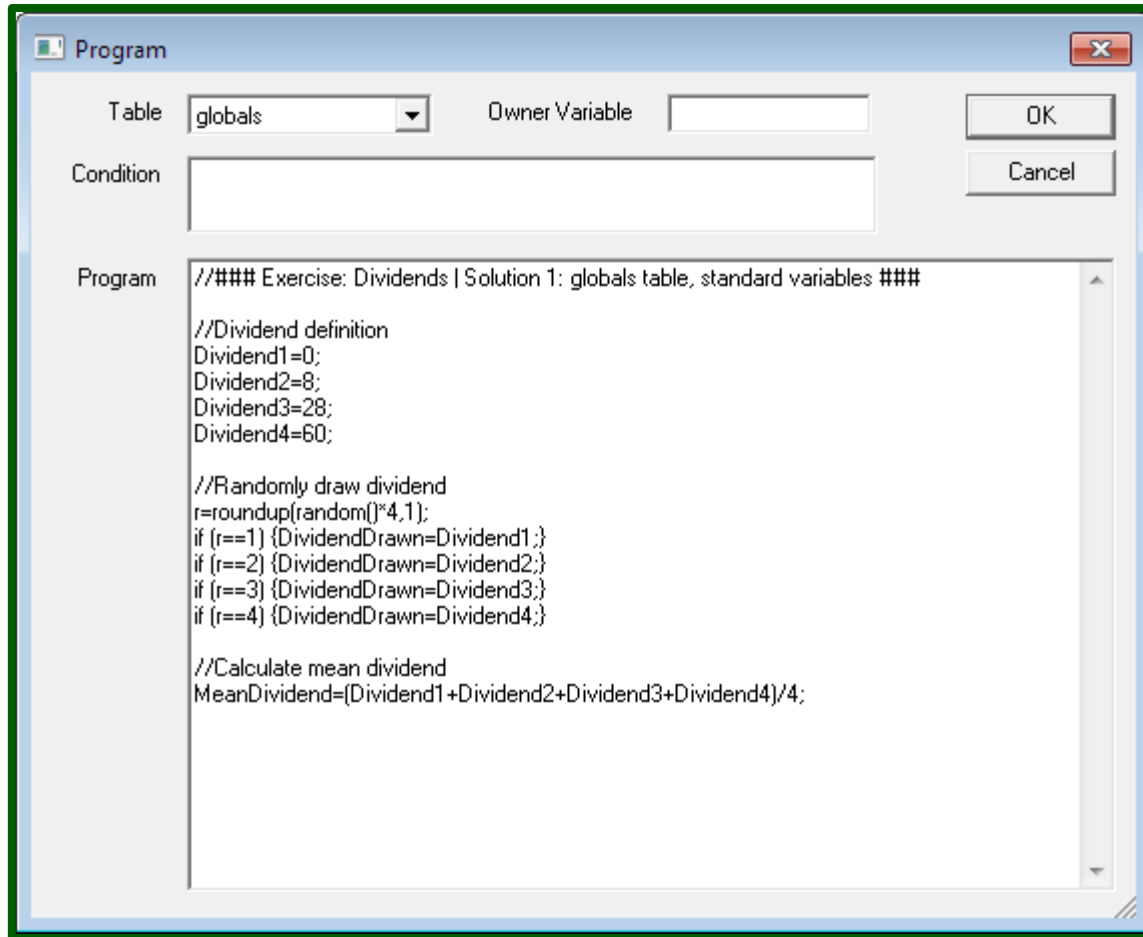
Dutch auction 2/2*

- The computer conducts a Dutch auction for 3 units of the good in each group
- The computer starts the auction at a price of 100 and counts down to 0, reducing the price by 1 every 3 seconds
- If a subject buys the good for the current price, the auction for the next good starts, until all goods are sold
- At the end, subjects see their profit, which is:
 - If they bought a good: The difference between their private valuation of the good and their purchase price
 - If they bought no good: Zero
 - Subjects may not buy at a price higher than their valuation

* Adapted from lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

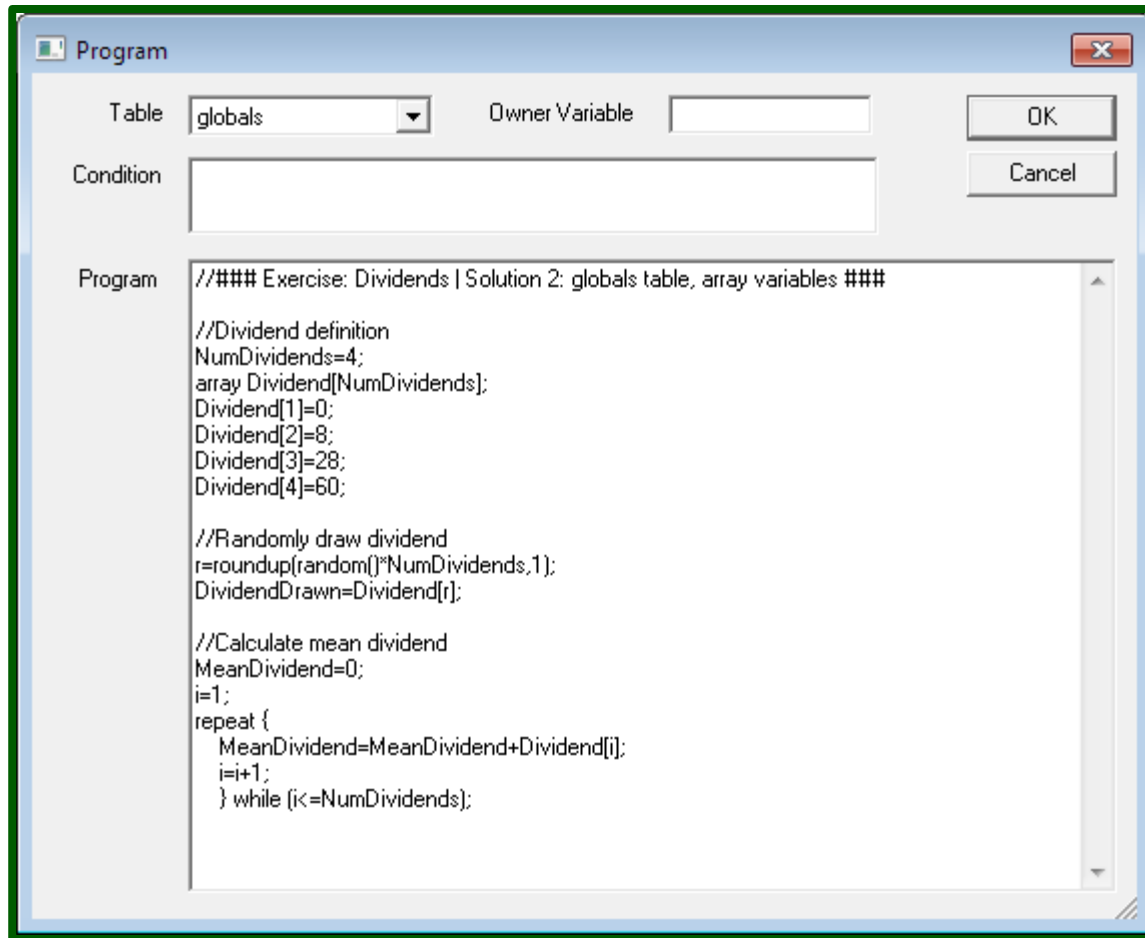
- Task: Create table/variable structure to save four possible dividend amounts
 - Dividend amounts: 0, 8, 28, 60
 - Allow for randomly picking a dividend amount and saving it in DividendDrawn
 - Allow for calculation of mean dividend and saving it in MeanDividend
 - Extra: Think about steps necessary to extend scenario to 8, or a variable number of possible dividend amounts

- Solution 1: globals table, standard variables

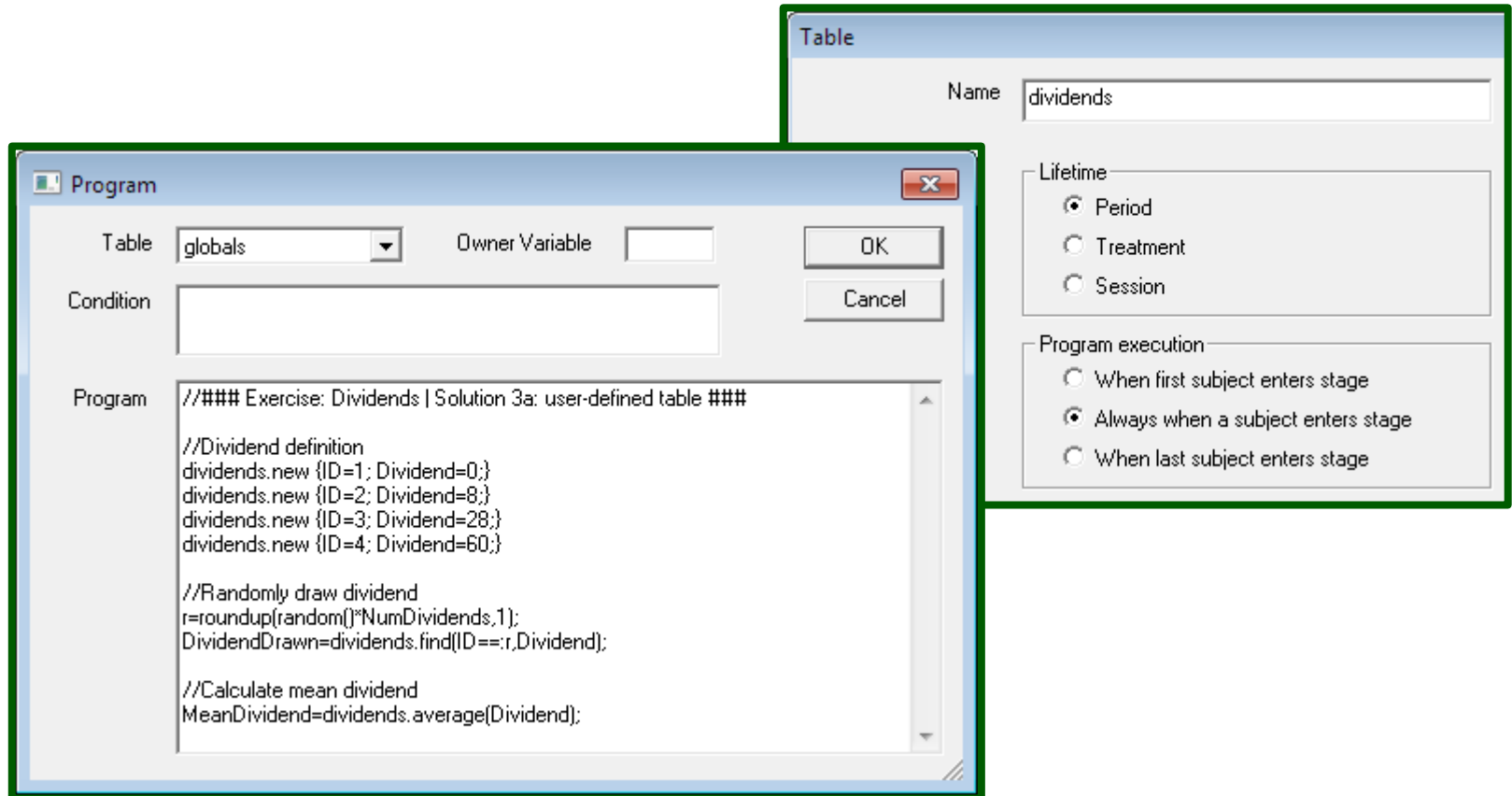


Dividends 3/5

- Solution 2: globals table, array variables



- Solution 3a: user-defined table



- Solution 3a: user-defined table with table loader

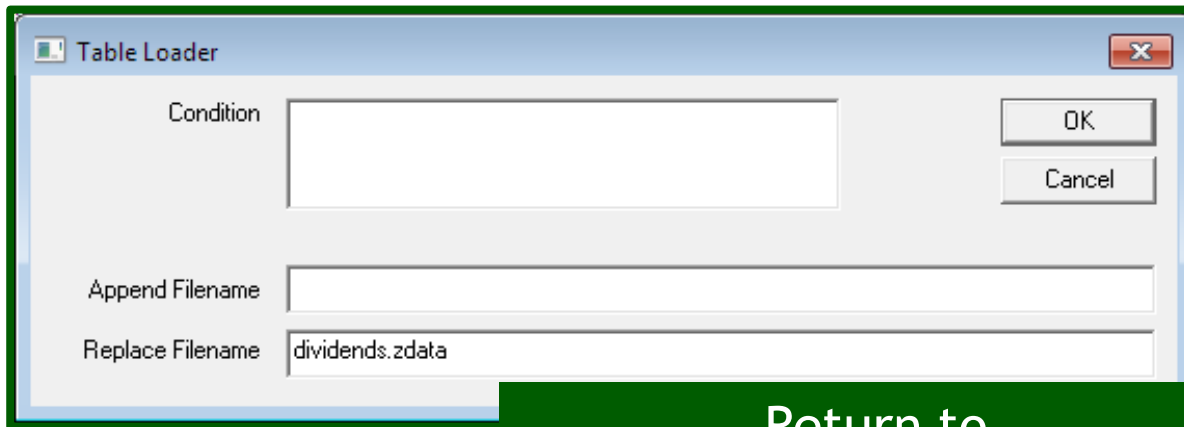


Table Loader

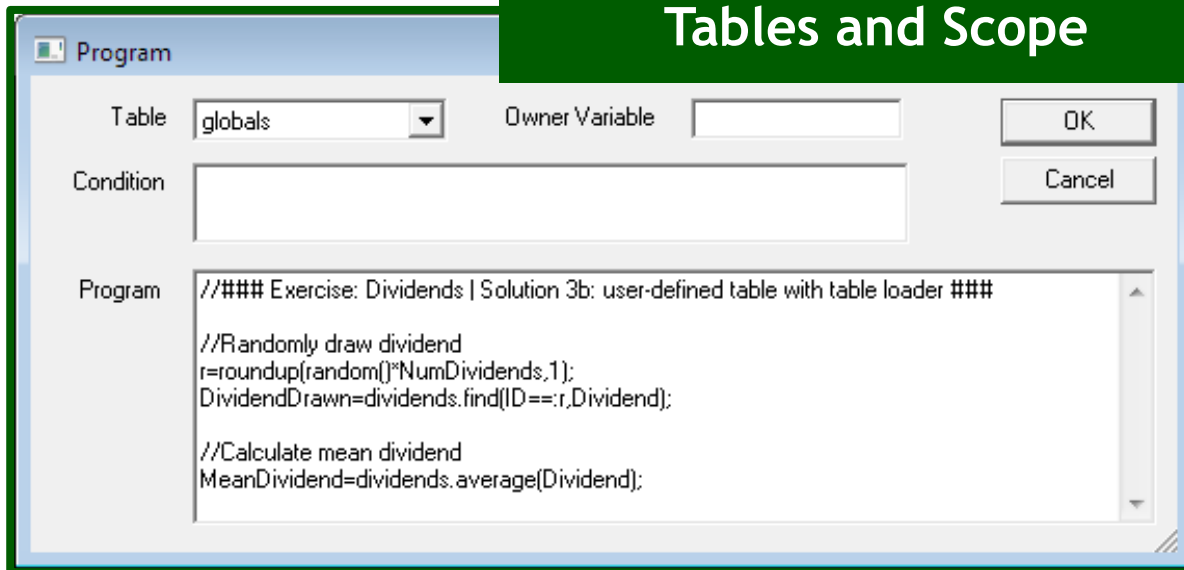
Condition

Append Filename

Replace Filename

OK Cancel

Return to
Tables and Scope



Program

Table

Owner Variable

Condition

Program

```
##### Exercise: Dividends | Solution 3b: user-defined table with table loader #####  
  
//Randomly draw dividend  
r=roundup(random()*NumDividends,1);  
DividendDrawn=dividends.find(ID==:r,Dividend);  
  
//Calculate mean dividend  
MeanDividend=dividends.average(Dividend);
```

OK Cancel

Random ending period

- Task: Experiment which ends with $1/6$ probability in every period beginning with period 10
 - Set periods to 1 in background
 - Set header not to display total number of periods
 - Have RandomEnding stage at end of program
 - Set Participate = if(Period > 9, 1, 0)
 - In this stage, throw a die and enter DieThrow on experimenter z-leaf or on code-locked z-leaf of subject 1 (if no experimenter subjects)
 - Set RepeatTreatment = if(Period < 10 | DieThrow < 6, 1, 0)

Return to
Subject Progression

Progression within a period

- Groups of 4: 3 firms, 1 investor

[Return to
Subject Progression](#)

- Structure:

- 
- Firms and investor discuss accounting standard
 - Firms vote on accounting standard (unanimous)
 - If necessary, firms vote again
 - Firms decide on report
 - Investor decides on valuation
 - Experimenter throws die for investor results
 - Firms and Investors receive results

[See CodeProgression.ztt](#)

- Groups progress through stages at different speeds
- If groups are fixed, they may even progress through “periods” at different speeds

A Comprehensive Introduction to z-Tree

Stefan Palan

ztree@palan.biz

<http://academic.palan.biz>



Call Auction

- Text

Market mechanism comparison

- CDA

- Immediate execution
- Immediate feedback
- Little price uncertainty with market orders
- Faster reaction upon new information

- CA

- Price uncertainty when submitting orders
- „Better“ price
 - Less noisy
 - Better information aggregation
- Less risk for uninformed traders
- Less chance for manipulation

Sorting values in z-Tree

- Example: Call Auction
- Program in globals table aggregates offers in contracts table into price/volume list - sorted ascending in price - in pricelist table:

```
if (contracts.count(p>0)>0) {  
  //Writes minimum and maximum price into variables in the globals table  
  minprice=contracts.minimum(same(Period),p);  
  maxprice=contracts.maximum(same(Period),p);  
  //Sets variable price equal to the lowest price in this period  
  price=minprice;  
  //Loops through all prices offered in this period and writes possible purchase and sales volume  
  //into pricelist table  
  n=1;  
  repeat {  
    //Creates new entry in pricelist table  
    pricelist.new {  
      //Entry consists of price, volume that could be sold at this price, volume that could be bought  
      //at this price, the total possible transaction volume at this price, and a counter variable  
      p=:price;  
      sellvol=contracts.sum(same(Period)&p<=:p&q<0,-q);  
      buyvol=contracts.sum(same(Period)&p>=:p&q>0,q);  
      vol=min(sellvol,buyvol);  
      n=:n;}  
    n=n+1;  
    //Increments the price to the next offered price or to a price>maxprice if maxprice has been reached  
    if (price<maxprice) {price=contracts.minimum(same(Period)&p >\price,p);} else {price=maxprice+1;} } while  
(price<=maxprice);  
    //Sets price equal to the mean of the prices which produce maximum volume  
    price=pricelist.average(same(Period)&vol==pricelist.maximum(same(Period),vol),p);}
```

Auxiliary Programs

- ABTutor
- hroot
- ORSEE
- Stata