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

Structuring a typical session 2/3

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.

Structuring a typical session 3/3

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

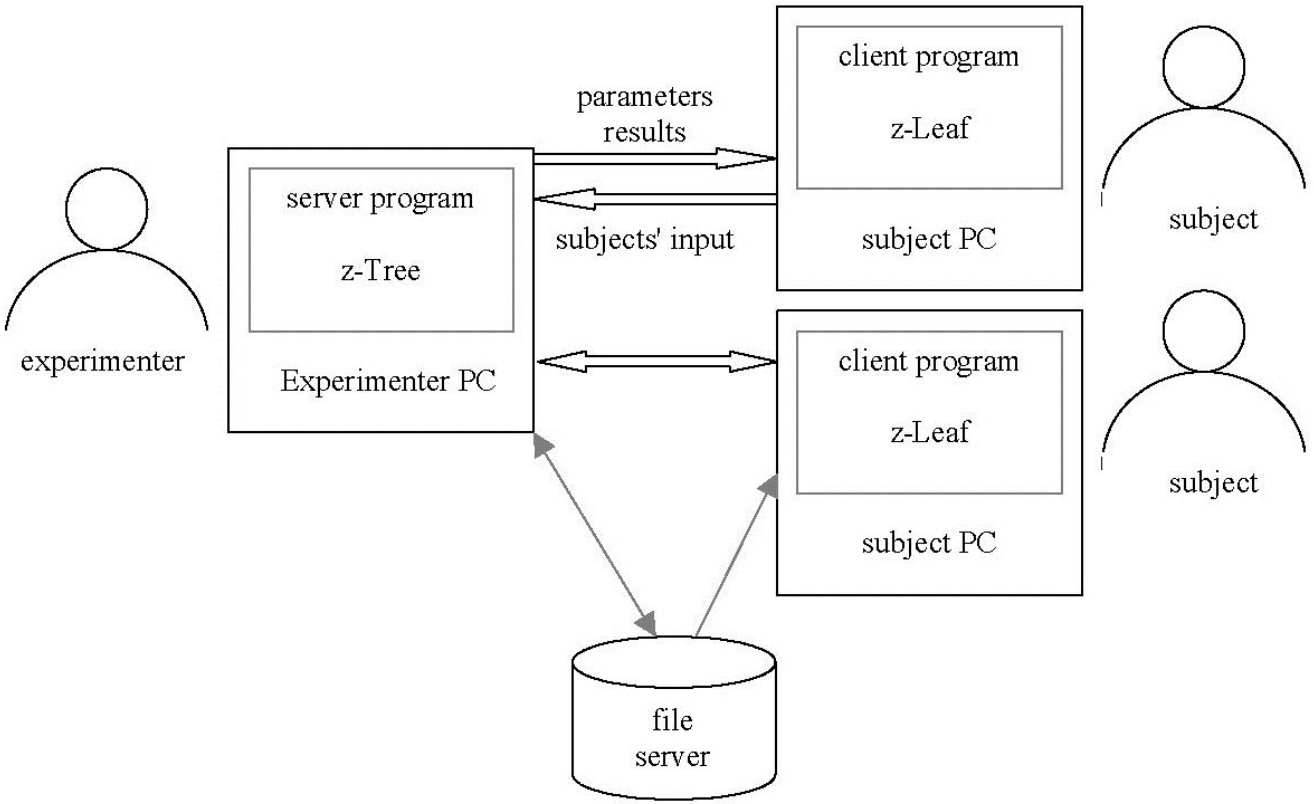
- Follow session structure cheat sheet diligently
  - Contains procedural steps
  - Contains program settings
  - Contains required material (dice, cards, etc.)
- Note special occurrences in experimenter diary
- Backup everything

Characteristics and features

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



Source: Tutorial, p. 6.

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
    - `/xlsdir`, `/sbjdir`, `/adrdir`, `/paydir`, `/gsfdir`, `/tempdir`, `/datadir` (autosaves of .ztt and .ztq), `/leafdir` (server.eec)
    - e.g. `/adrdir c:\data`

Files used by z-Tree

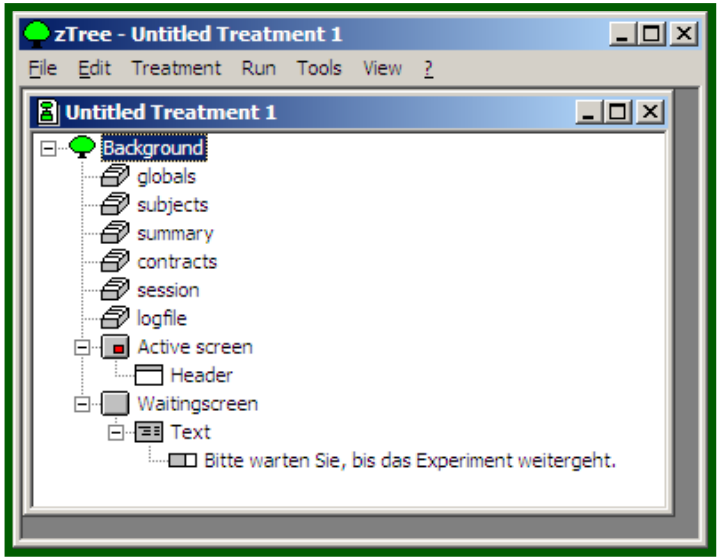
ztree.exe	Server program file
zleaf.exe	Client program file
name.ztt	Treatment code file
name.ztq	Questionnaire code file
name.txt	Parameter input file
@1.ztt, @2.ztt, … @1.ztq, @2.ztq, …	Backups of treatments and questionnaires
@db.txt, @lastclt.txt, @prevdb.txt	Temporary files
090330_0804.adr	Subject 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*

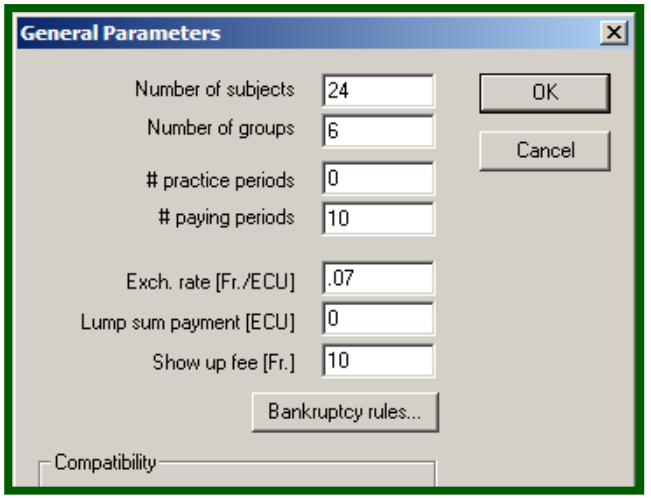
- Groups of n = 4 subjects
- Initial endowment of subject i: $w_i = 20$
- Contribution of subject i: $c_i \in [0, w_i]$
- Profit of subject i: $\pi_i = w_i - c_i + 1.6 \cdot \frac{1}{n} \cdot \sum_{j=1}^{4} c_i$
- New z-Tree treatment:



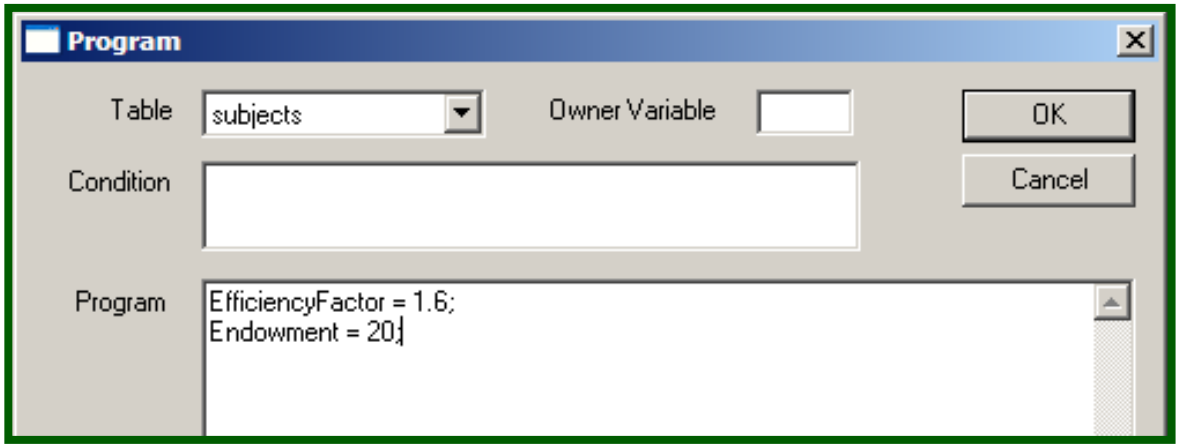
* Source: z-Tree Tutorial, sections 2.2-2.3.

Define parameters

- Define general parameters in Background:

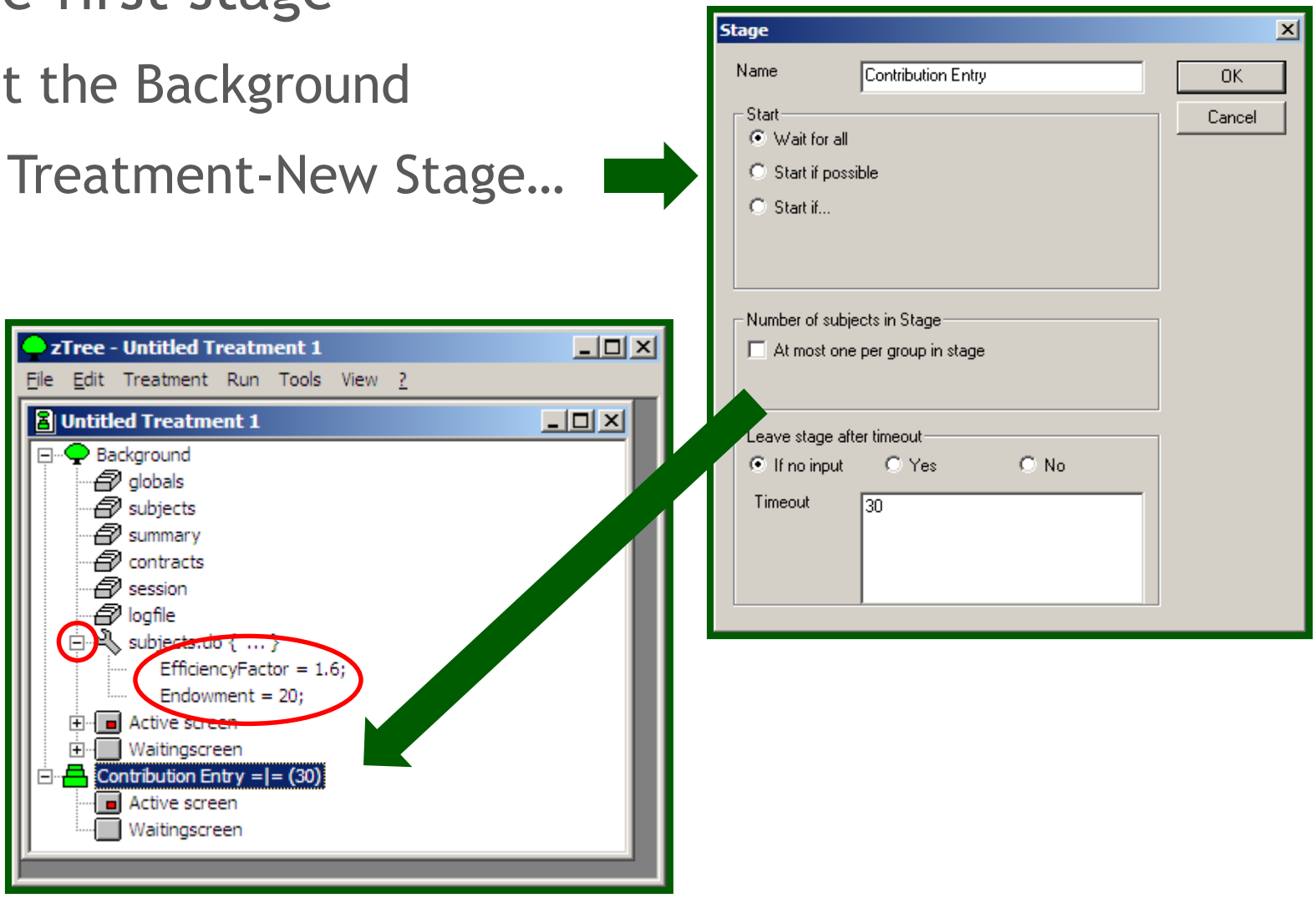


- Create new program
  - Select logfile in stage tree
  - Click Treatment-New Program…



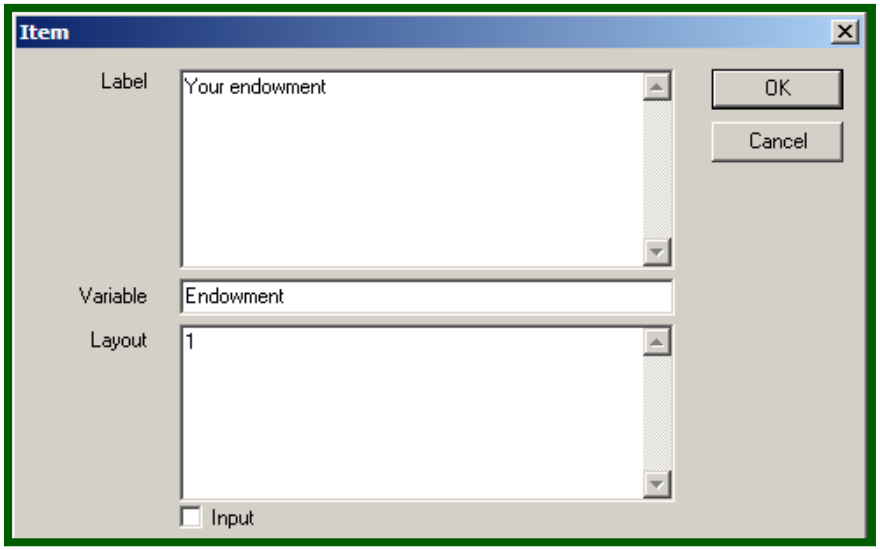
The first stage

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



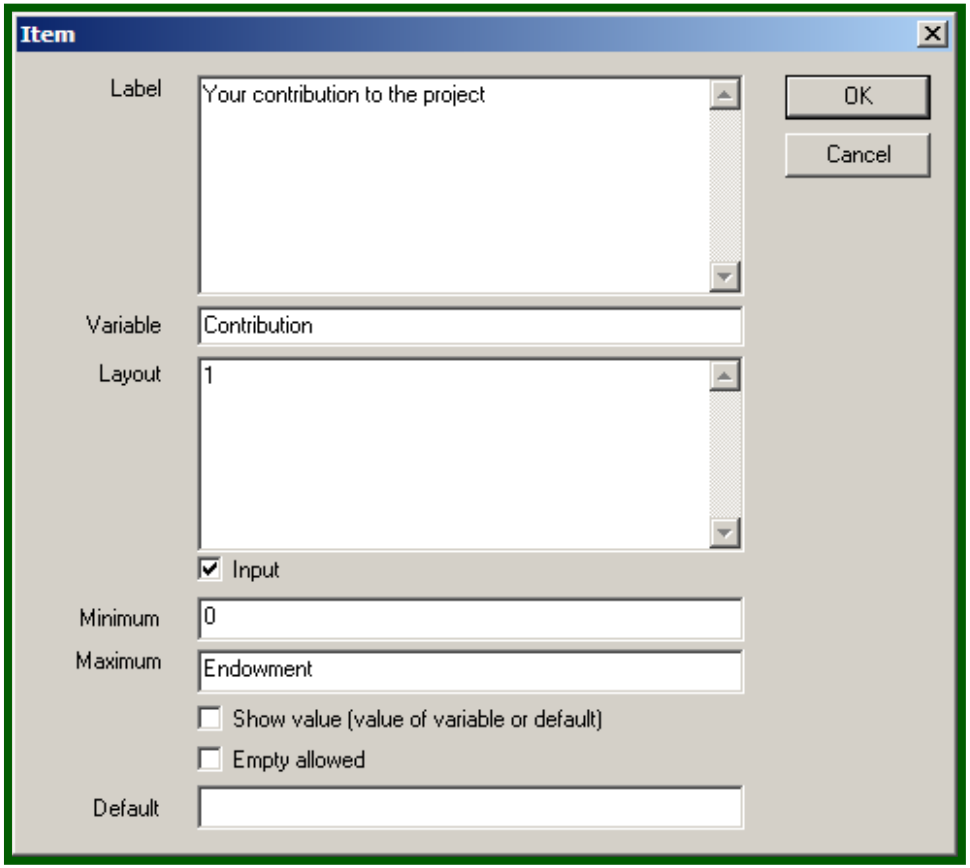
Designing the screen 1/2

- 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…



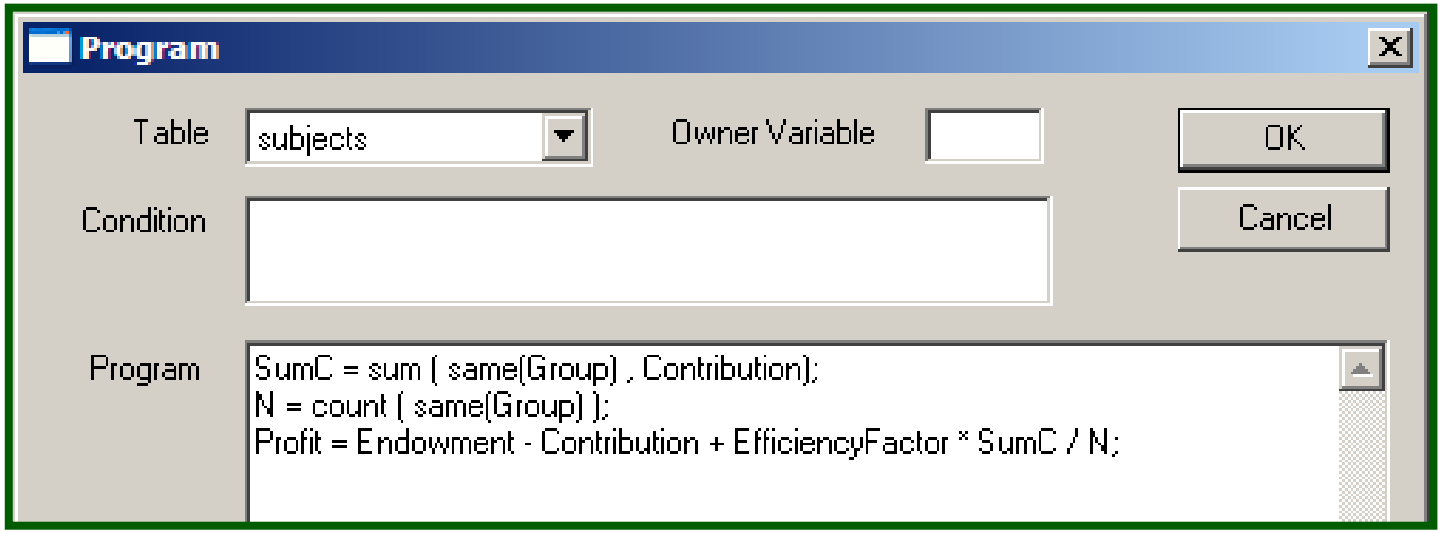
Designing the screen 2/2

- 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



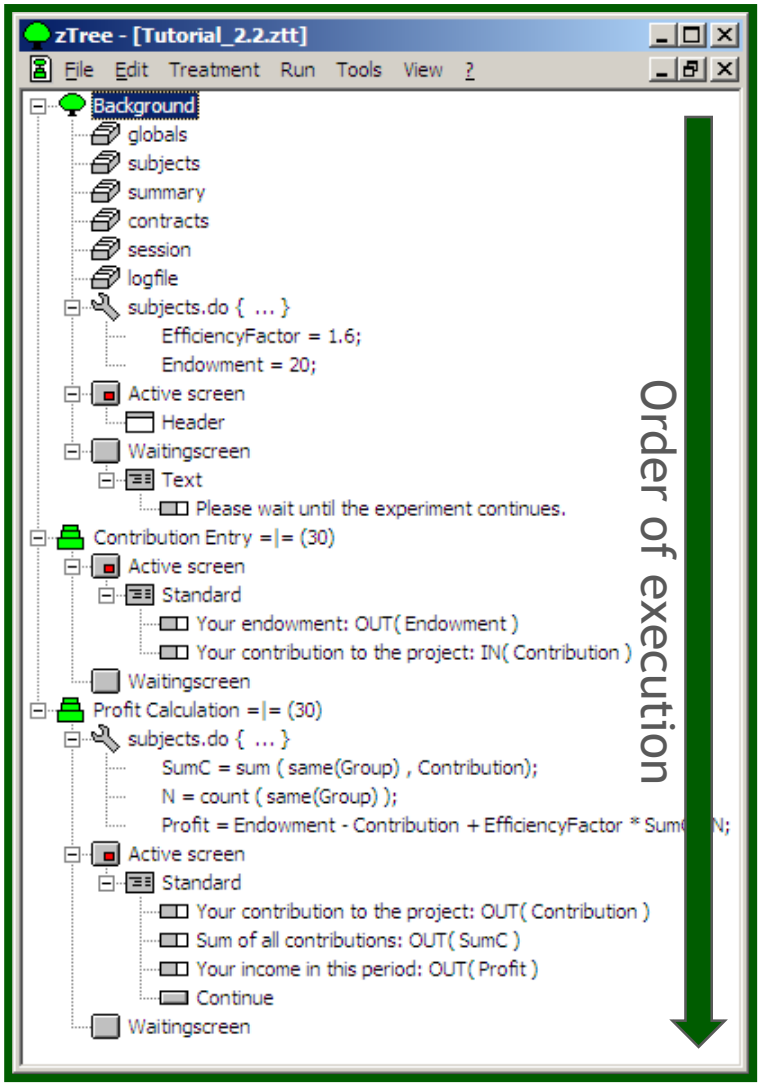
Profit calculation

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



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)

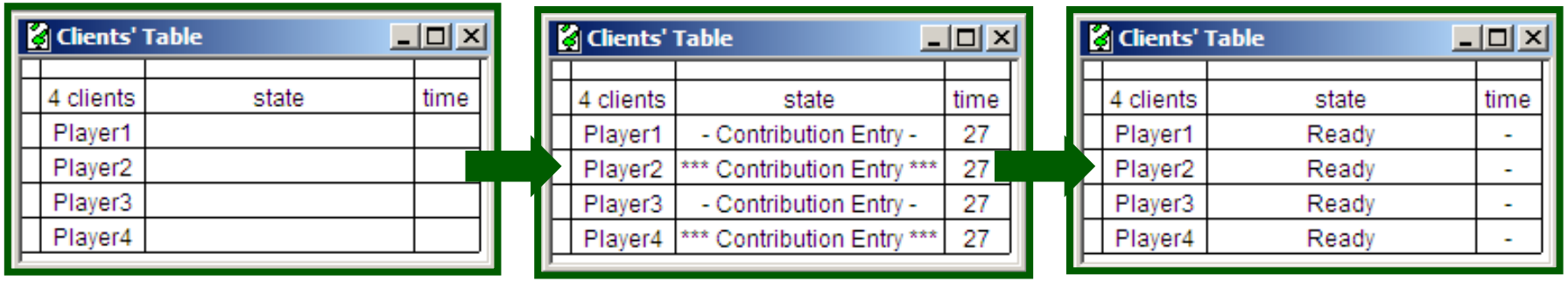


Starting multiple z-Leafs for testing

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

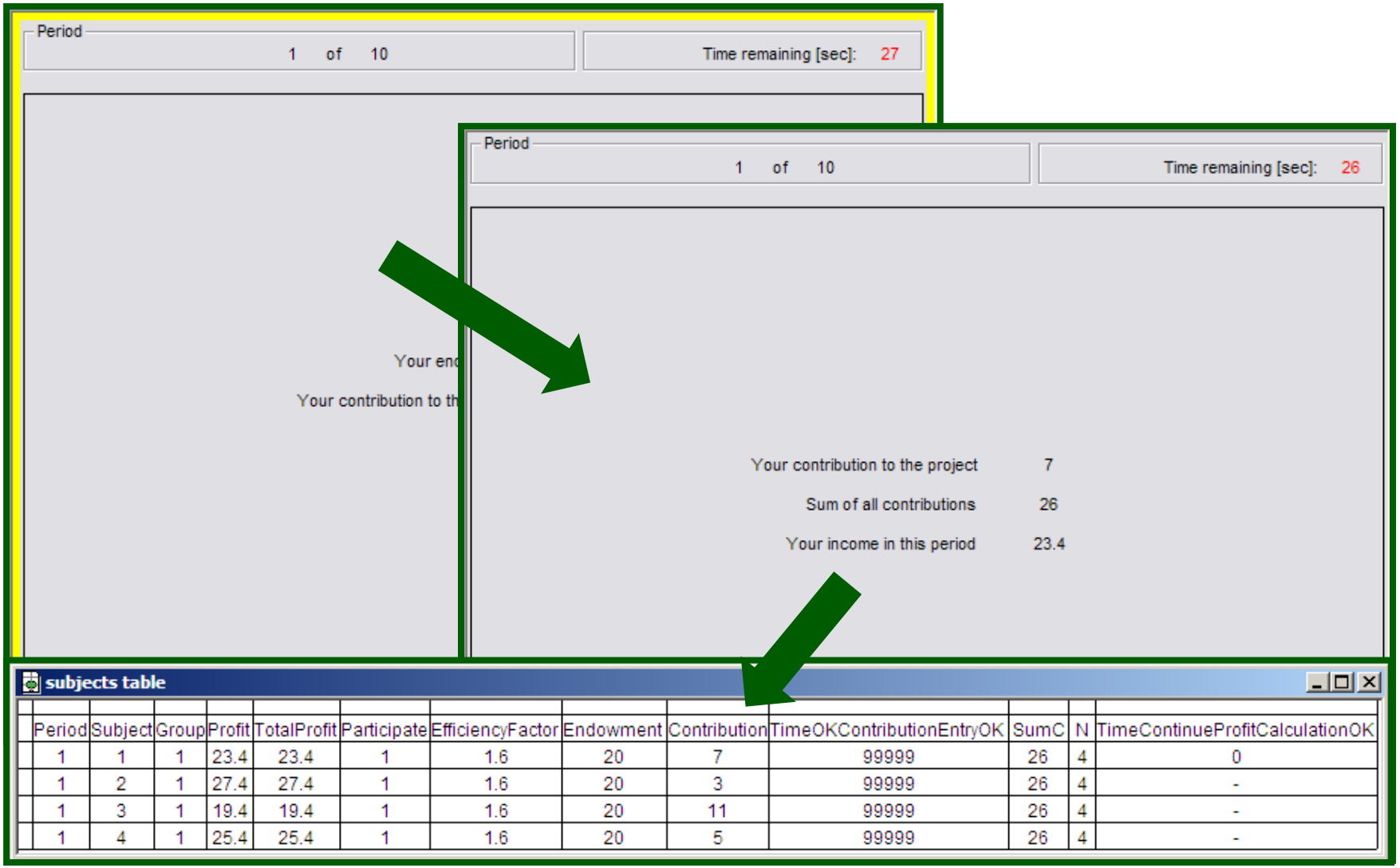


- Pause time using <F12>
- Resume time using <Shift>-<F12>
- Stop testing by clicking Run-Stop after this period

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	Subject	Group	Profit	TotalProfit	Participate	EfficiencyFactor	Endowment	Contribution	TimeOKContributionEntryOK	SumC	N	TimeContinueProfitCalculationOK
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	-

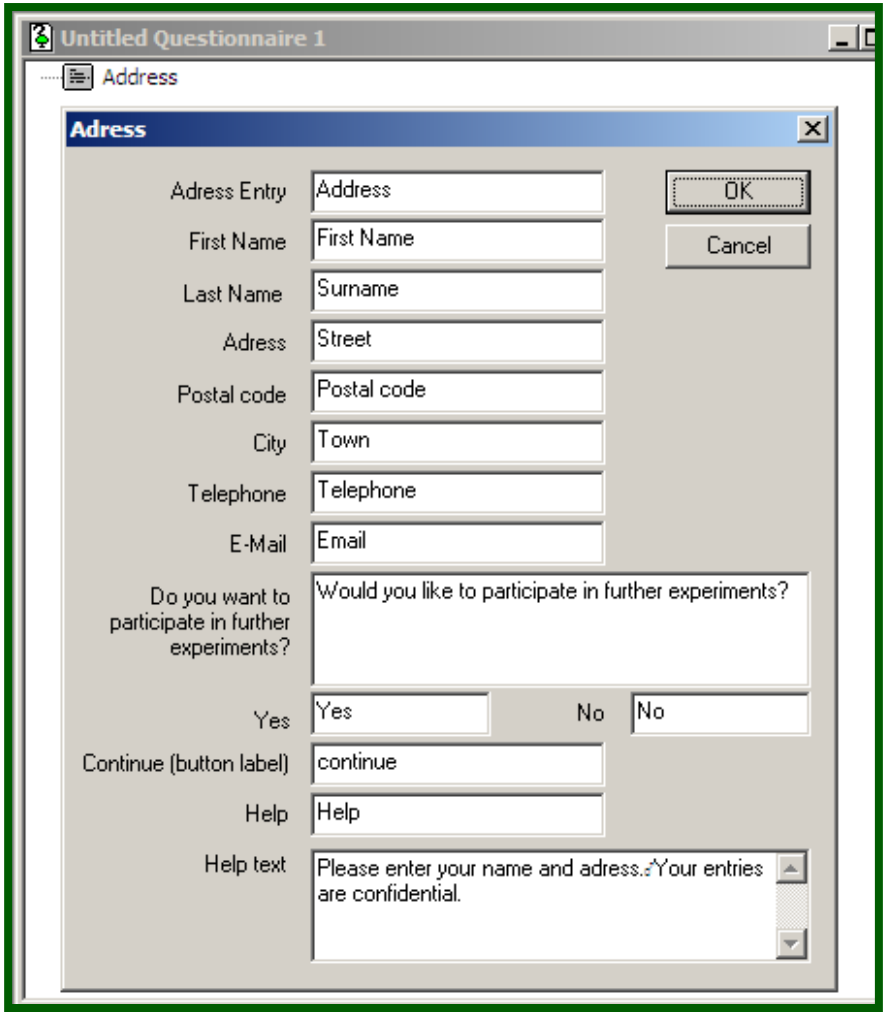
Resulting data output

- 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					

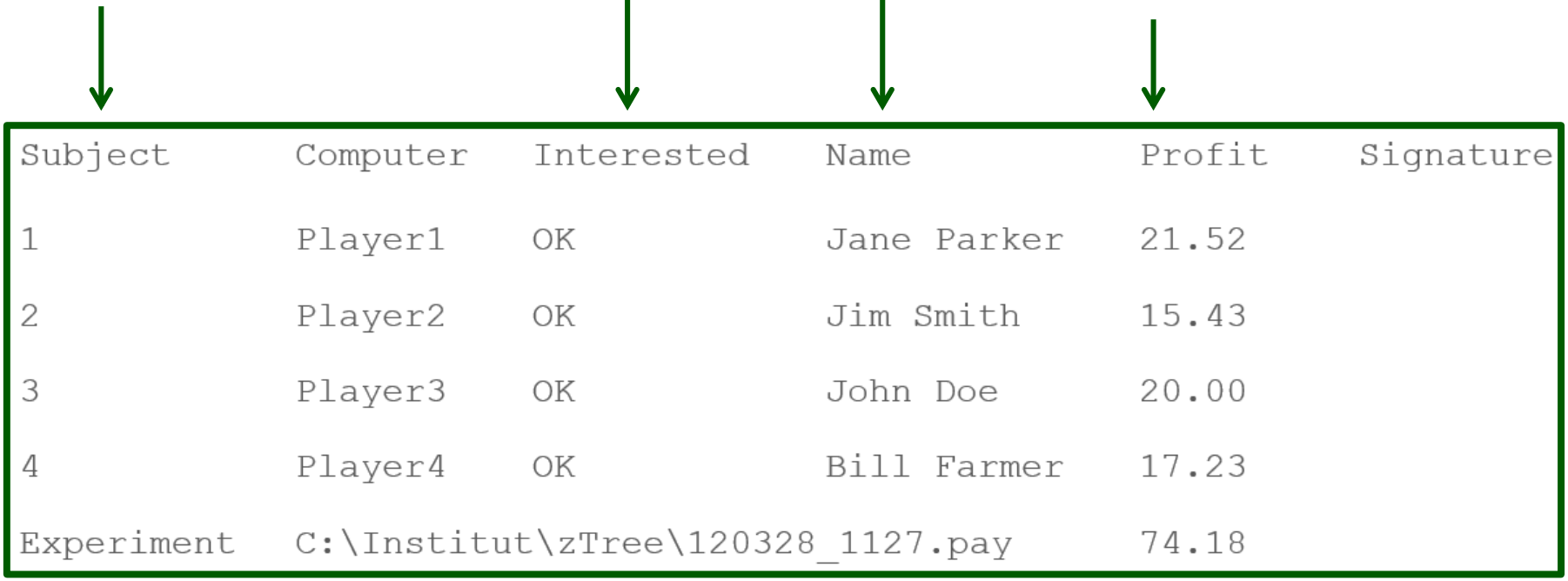
Creating new questionnaire

- 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



Payment file

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



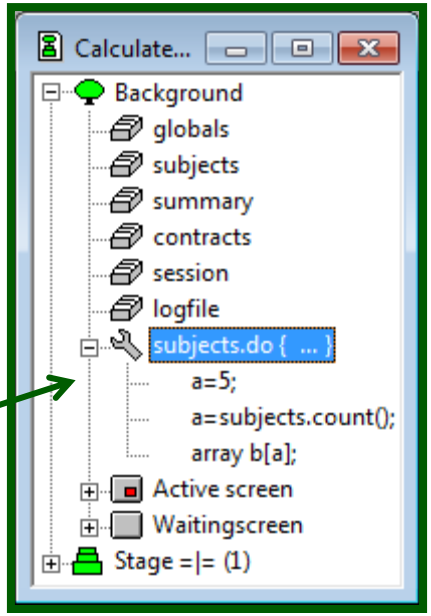
Subject	Computer	Interested	Name	Profit	Signature
1	Player1	OK	Jane Parker	21.52	
2	Player2	OK	Jim Smith	15.43	
3	Player3	OK	John Doe	20.00	
4	Player4	OK	Bill Farmer	17.23	
Experiment	C:\Institut\zTree\120328_1127.pay			74.18	

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”



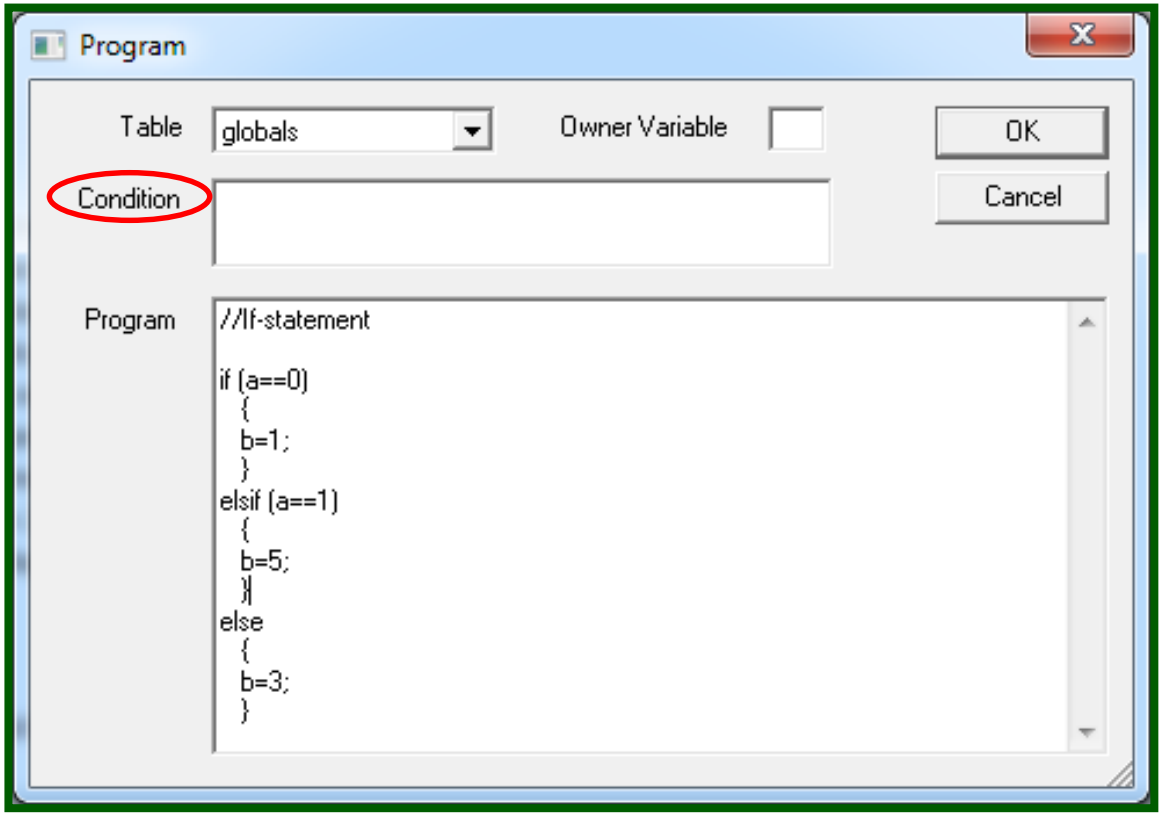
- String variables
  - New in Version 3.4.0 and later
    - b=“”;

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));

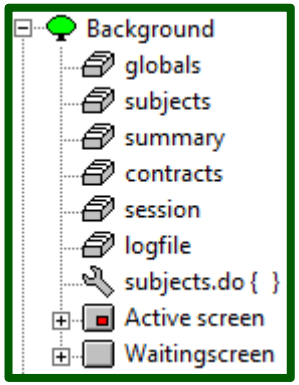
Conditional execution in z-Tree

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

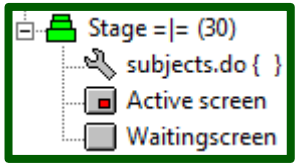


Structure of a treatment

- 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



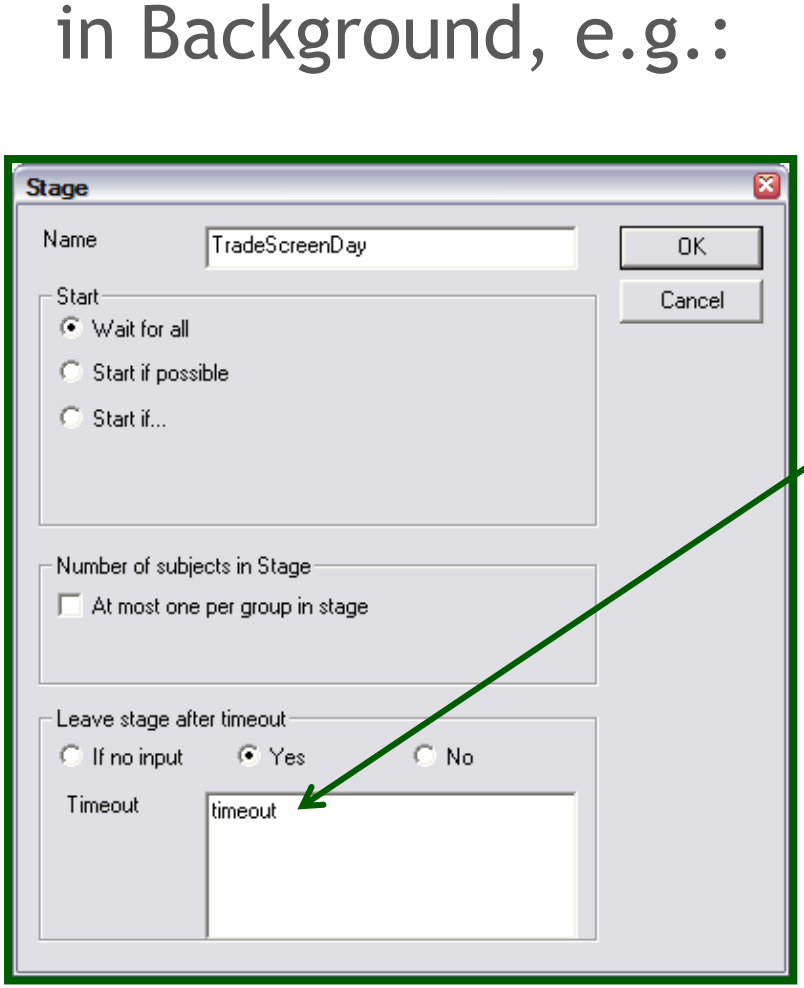
- Stages
  - Correspond to the “screens” displayed to subjects
  - Contain programs which run at the beginning of a stage

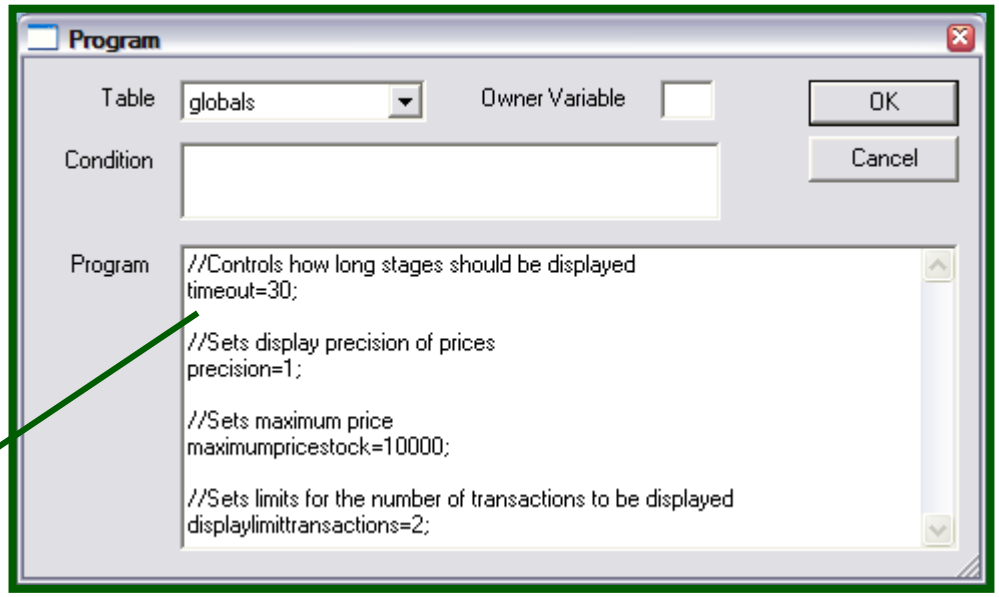


* Adapted from a z-Tree lecture by Maria Bigoni: http://www2.dse.unibo.it/bigoni/courses/ztree/Lecture_notes_handout.pdf, 31.08.2012, accessed on 19.12.2012.

Defining parameters in the Background

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

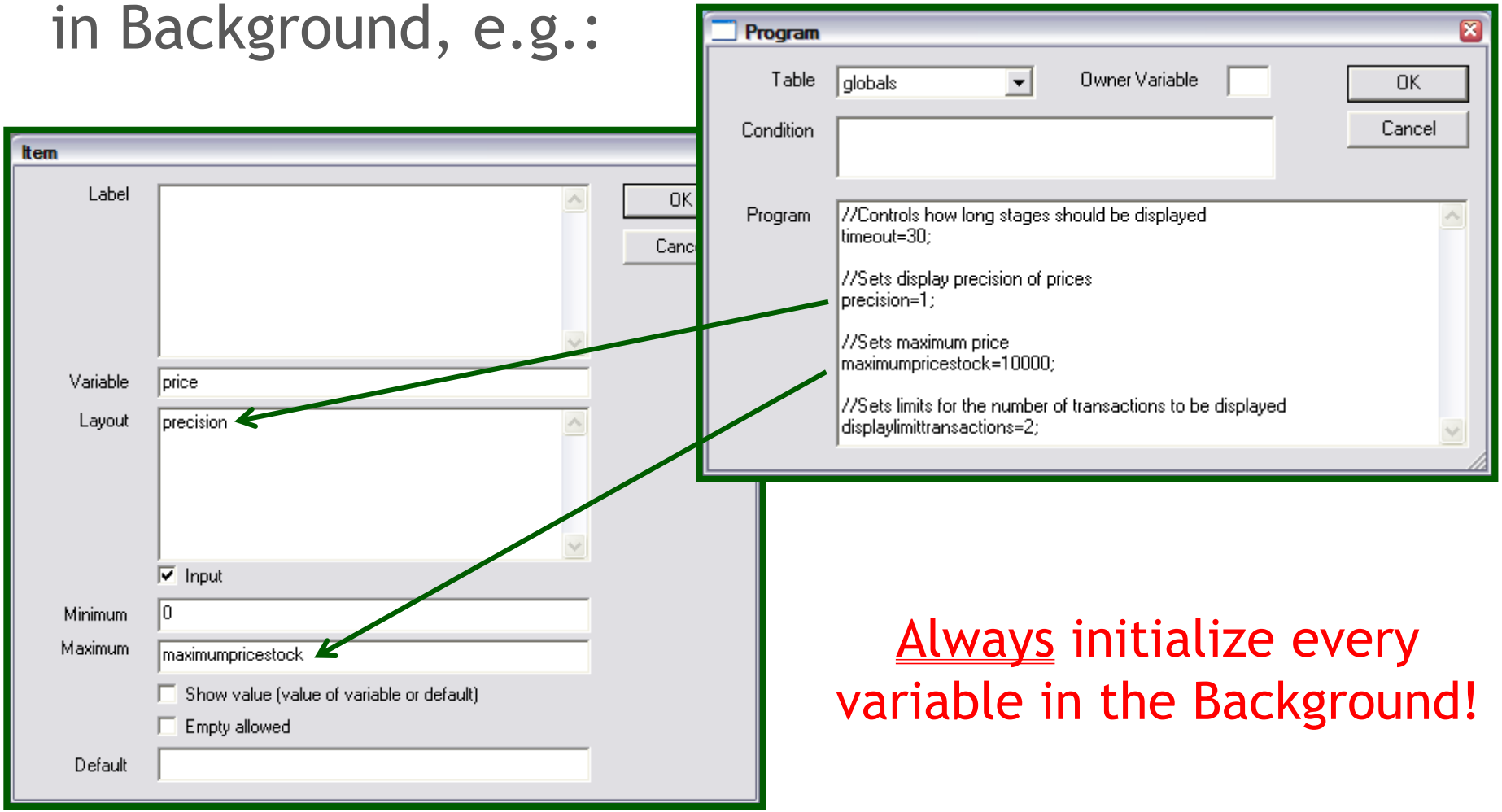




Always initialize every variable in the Background!

Defining parameters in the Background

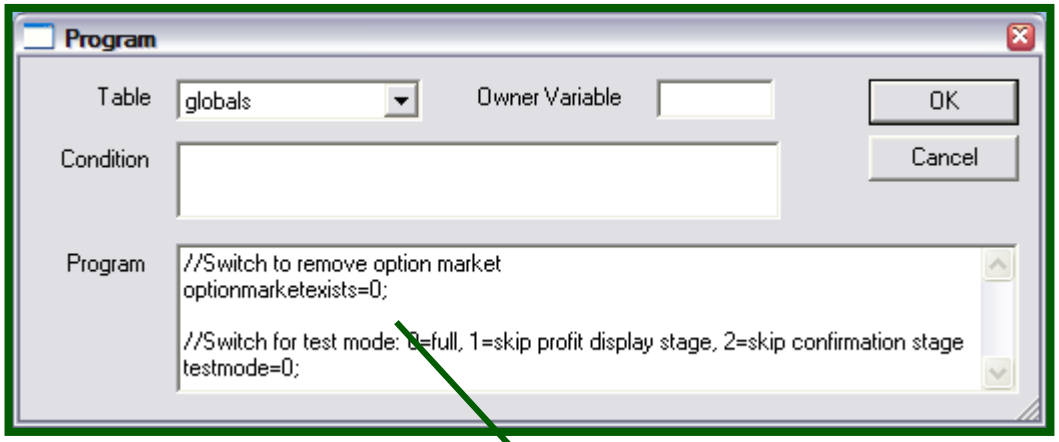
- Put general parameters into program in globals table in Background, e.g.:

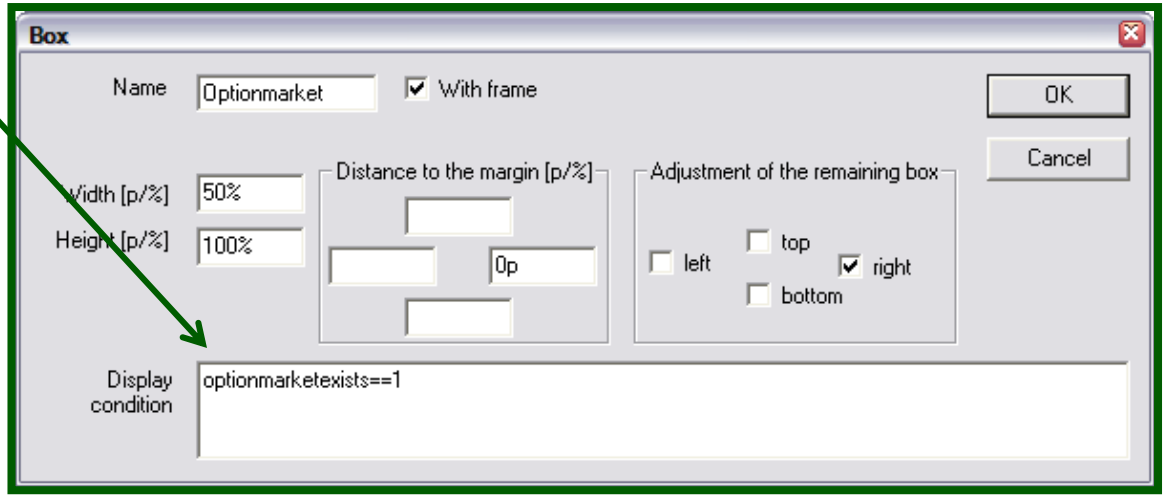


Always initialize every variable in the Background!

Defining „switches“ in the Background

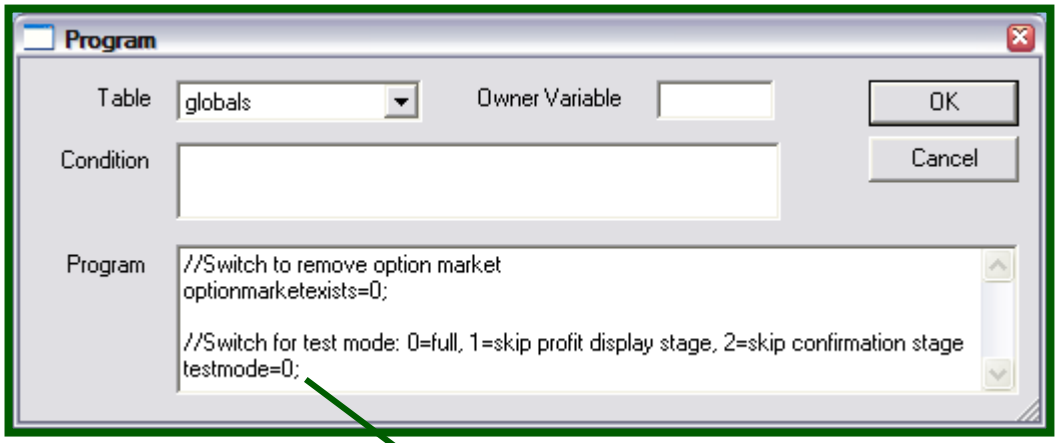
- Use variables to switch between test modes, and between different treatments within one .ztt file:

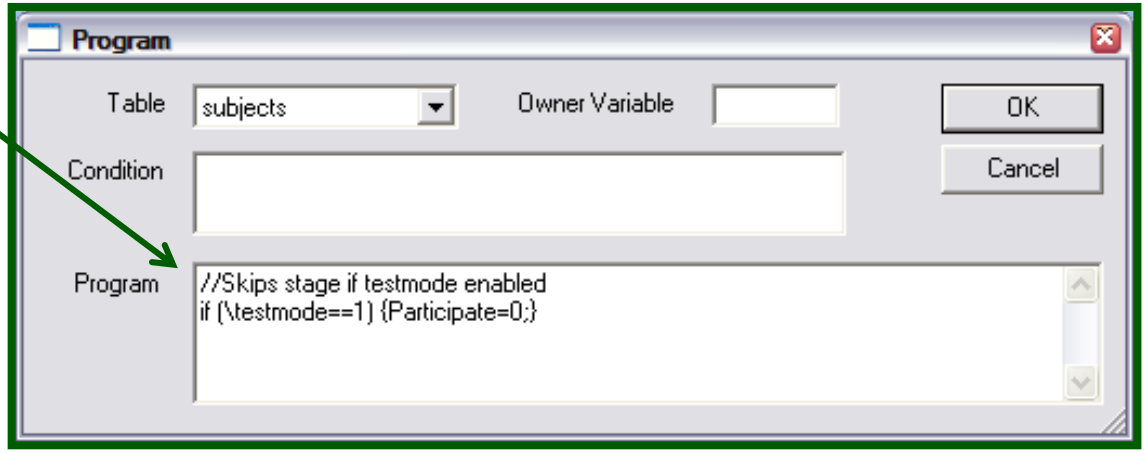




Defining „switches“ in the Background

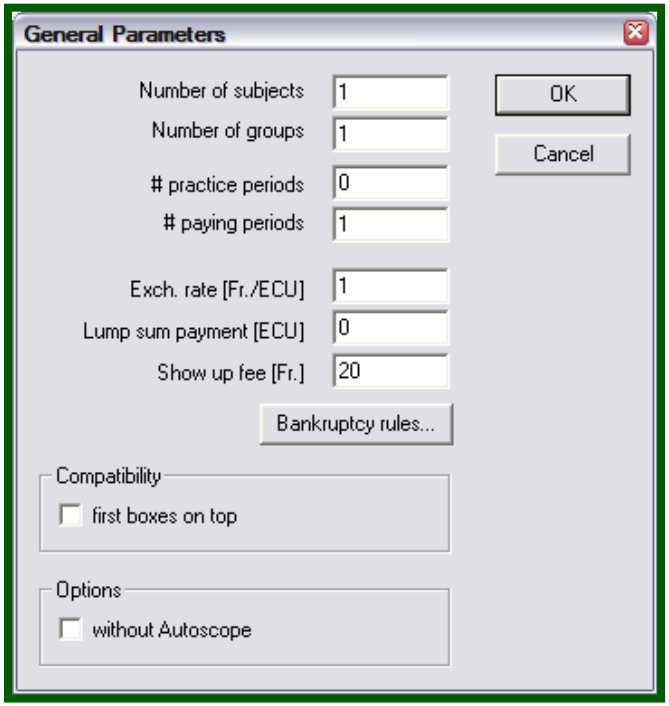
- Use variables to switch between test modes, and between different treatments within one .ztt file:





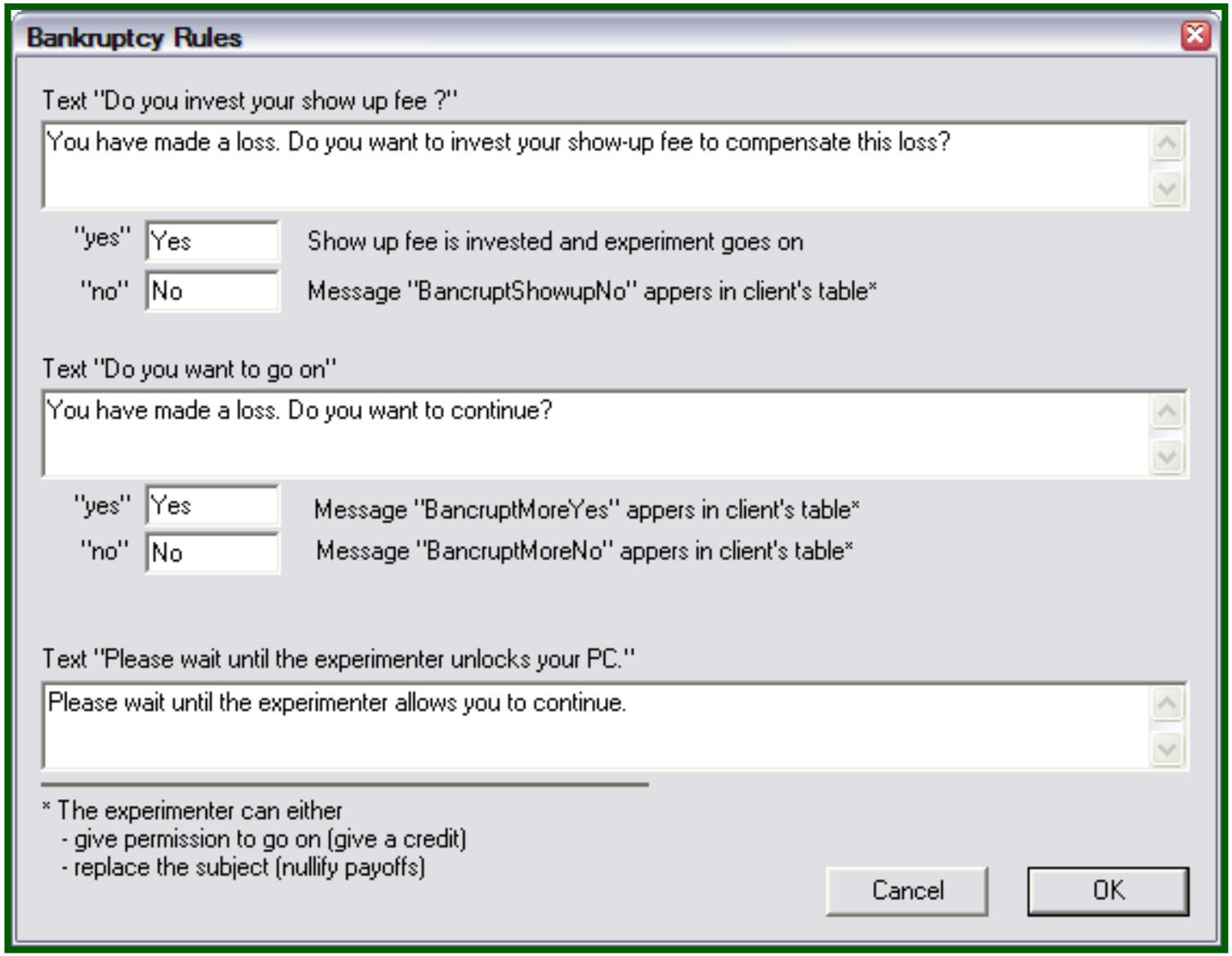
Calculation of payouts: Background

- # 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

Calculation of payouts: Bankruptcy



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

Calculation of payouts

Link to
Calculation of payouts
in
Good Practice and Advice

Creating a stage

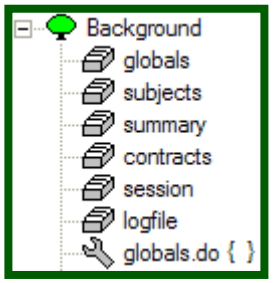
- Name is for documentation only; must be unique
- Start
- Timeout
- Active screen, Waitingscreen, Use background, Header
- Screen layout
  - Containers
  - Boxes
  - Absolute/relative size/position
  - Adjustment of the remaining box

Period structure

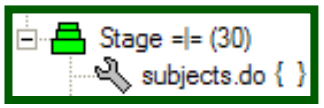
- Setting of standard variables
- Programs in the Background
- Specific parameter programs
- Role parameter programs
- Period parameter programs
- Programs at the beginning of a stage
- Programs in buttons (when clicked)
- Delayed sub-programs:
  later ( ) do { } / later ( ) repeat { }

Running programs

- Program in the background
  - Runs after the setting of standard variables

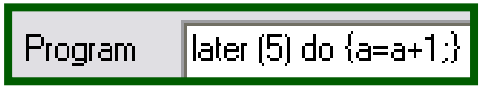


- Program in a stage
  - Runs at beginning of stage, before checking Participate

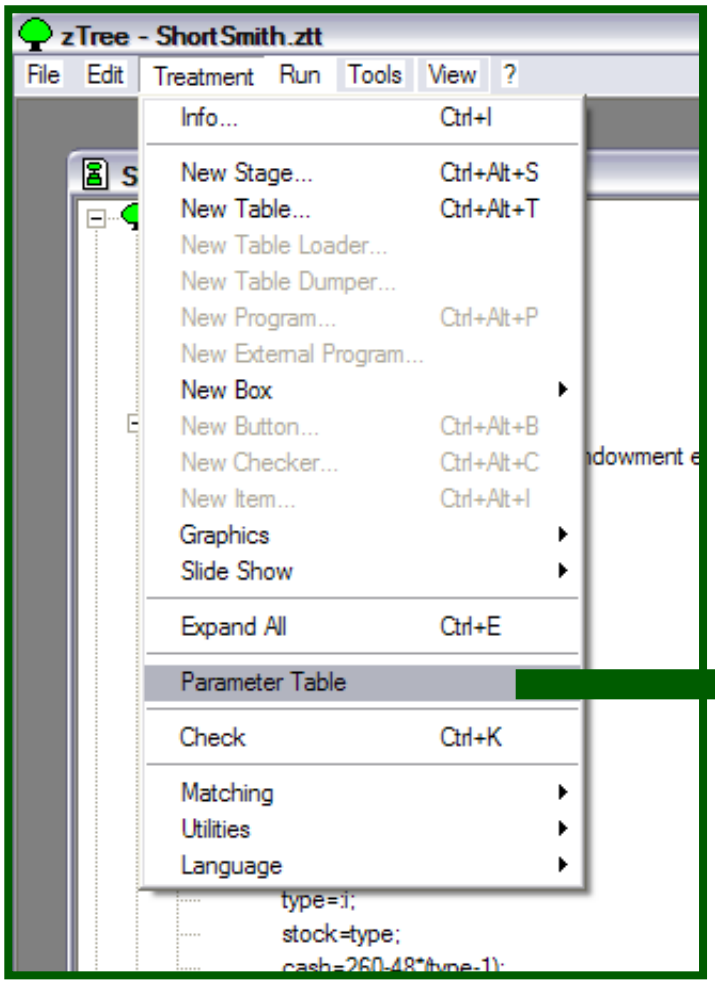


- Program in a button
  - Program in subjects table runs only for the subject pressing the button
  - Runs after checkers

- Program started by later ( ) do/repeat { } command
  - (Re-)Runs after pre-specified time

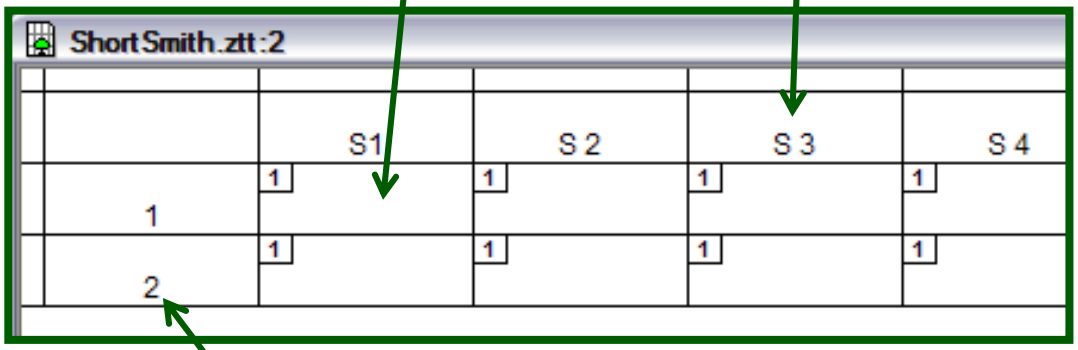


Parameter table



1. Specific parameters
   (after programs in the background)

2. Role parameters



3. Period parameters & prompt

Loops in z-Tree

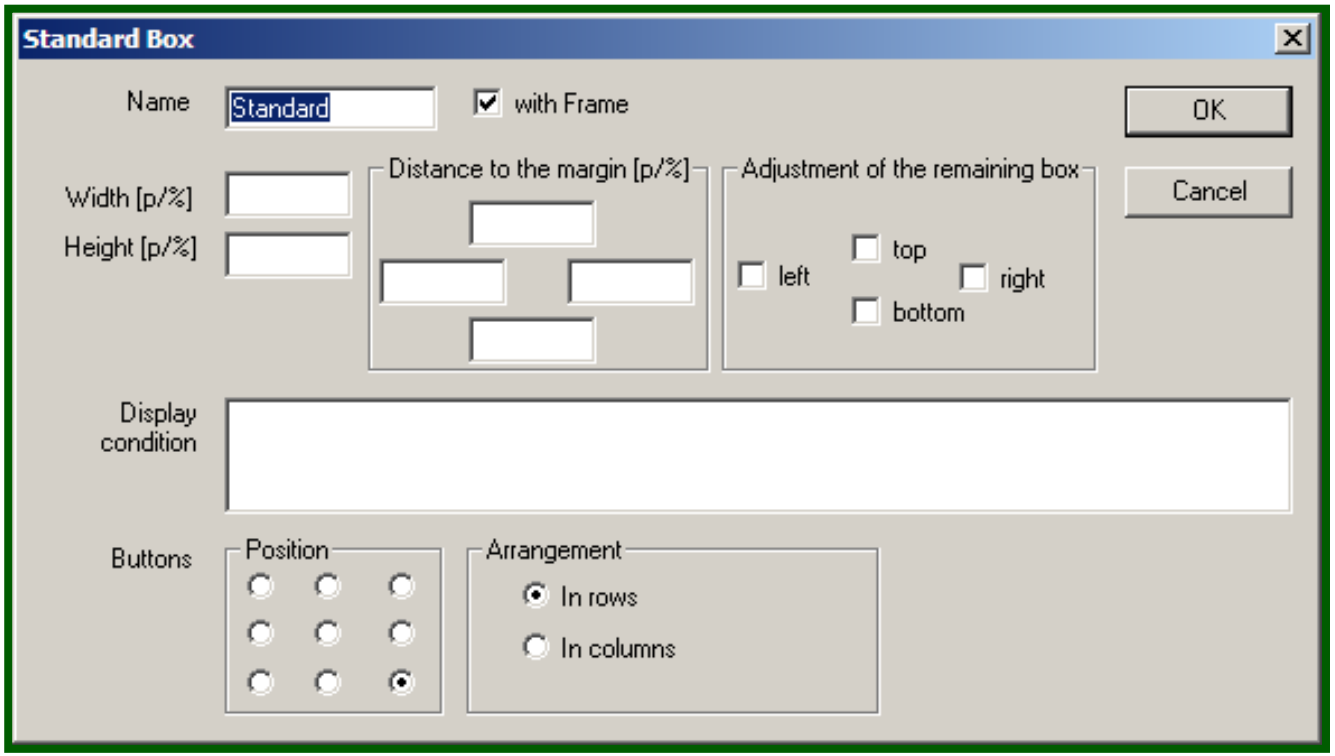
- Loops through the records of a table:
  - `contracts.do { player = :Subject; }`
- Loops using `while ( )`:
  - `while ( i <= 5 ) { i = i + 1; }`
- Loops using `repeat { } while ( )`:
  - `repeat { i = i + 1; } while ( i <= 5 );`
- Loops using `iterator( ).do`:
  - `iterator( i, 8, 12, 2 ).do { :player = 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

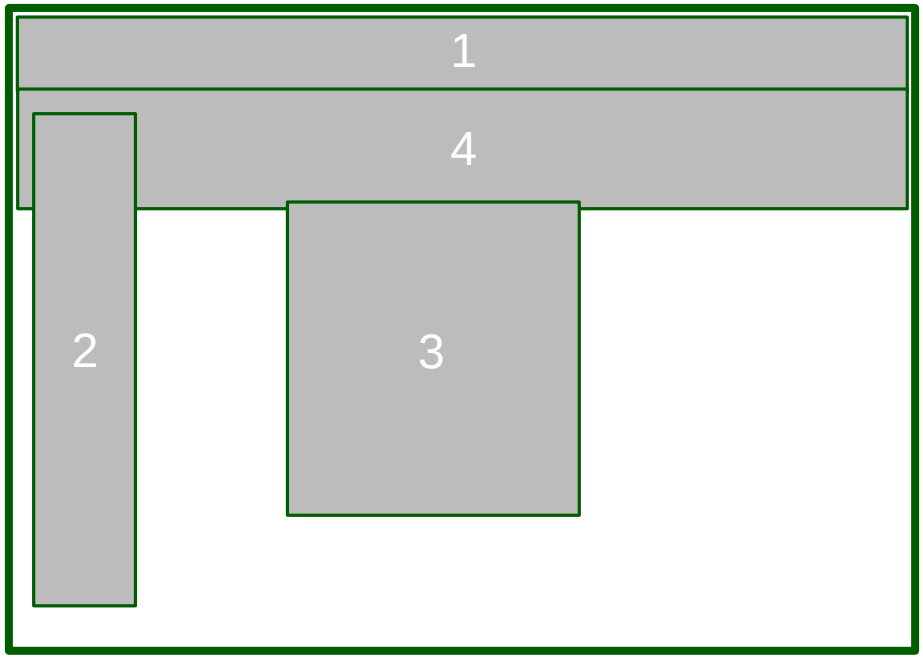
Screen layout

- Screen layout can be controlled in box dialog:



- See Demo_ScreenLayout.ztt for example layouts

Screen layout



Name 1 with Frame
Width [p/%]
Height [p/%] 10%
Distance to the margin [p/%] 0p
Adjustment of the remaining box: top

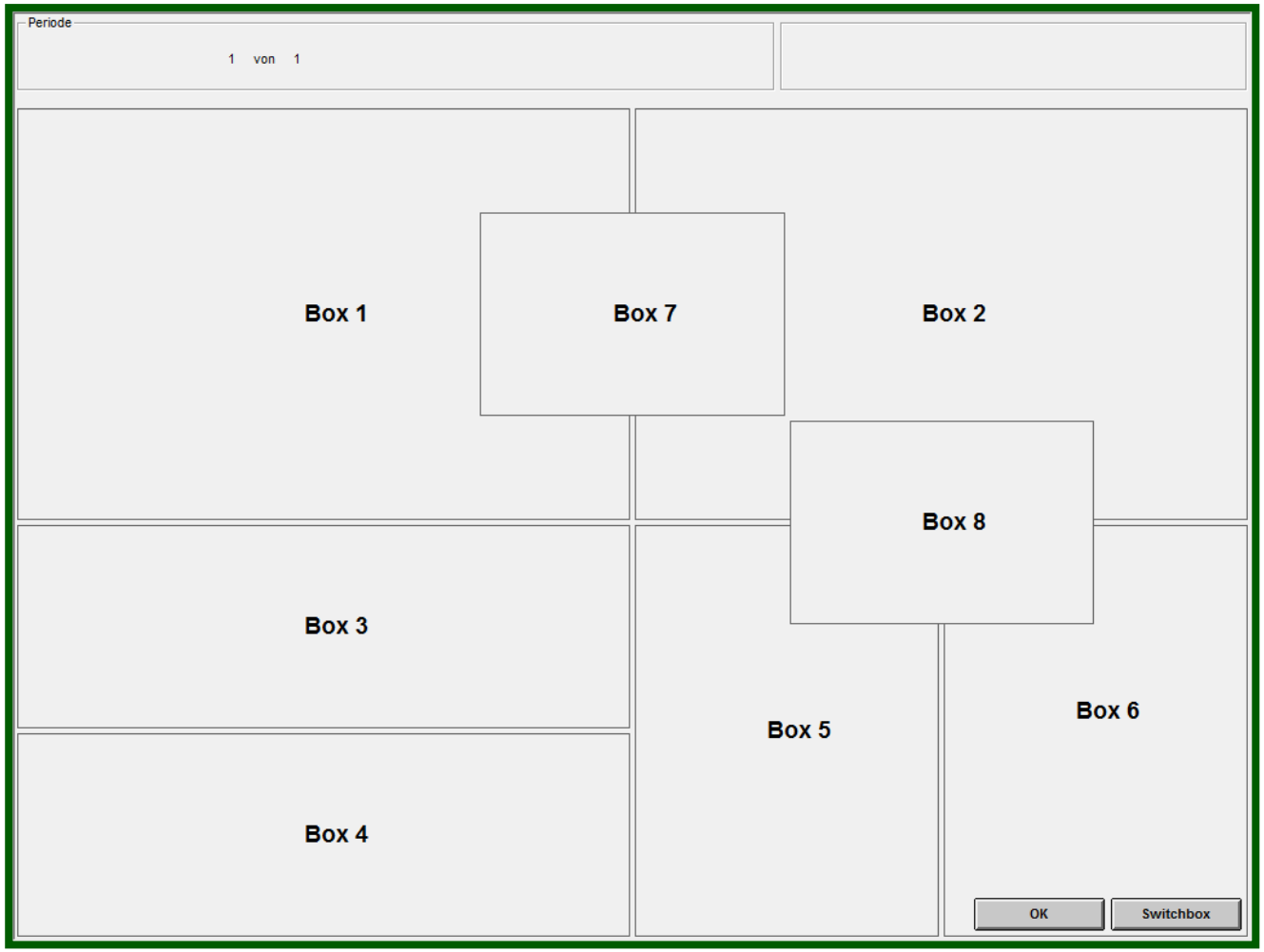
Name 2 with Frame
Width [p/%] 50p
Height [p/%]
Distance to the margin [p/%] 20p, 20p, 20p
Adjustment of the remaining box: left, top, right, bottom

Name 3 with Frame
Width [p/%] 30%
Height [p/%] 50%
Distance to the margin [p/%]
Adjustment of the remaining box: bottom

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

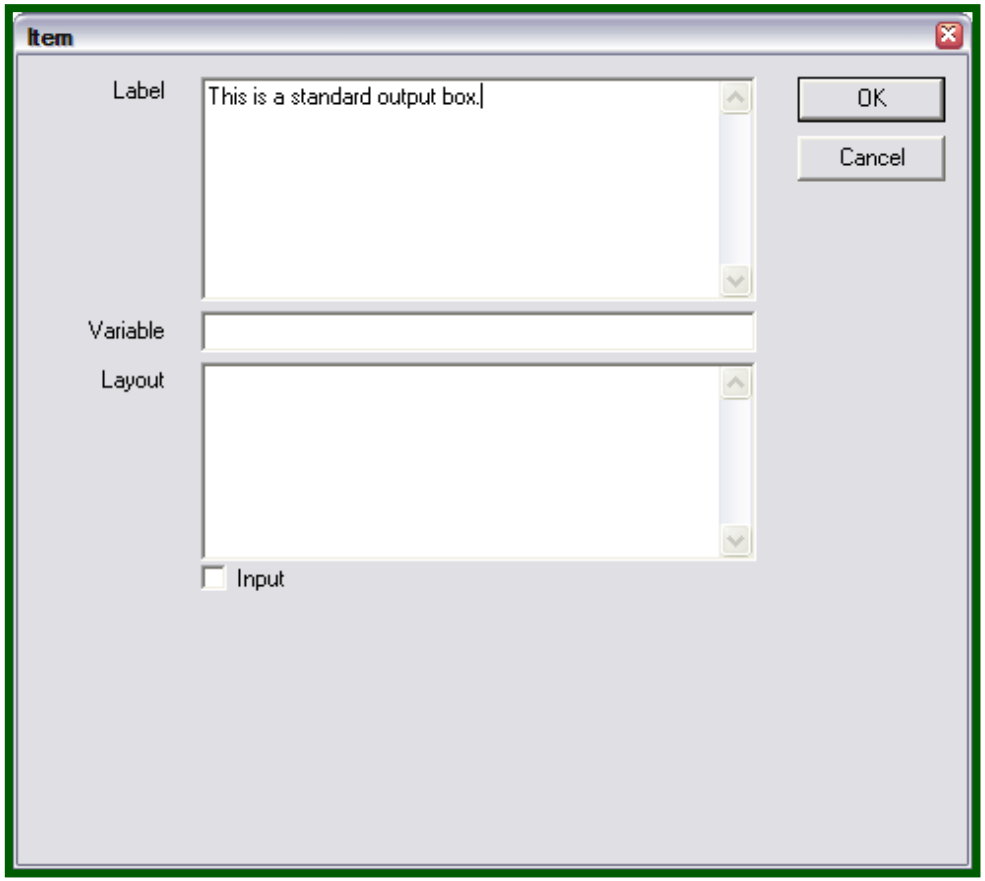
Exercise - Screen layout*

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



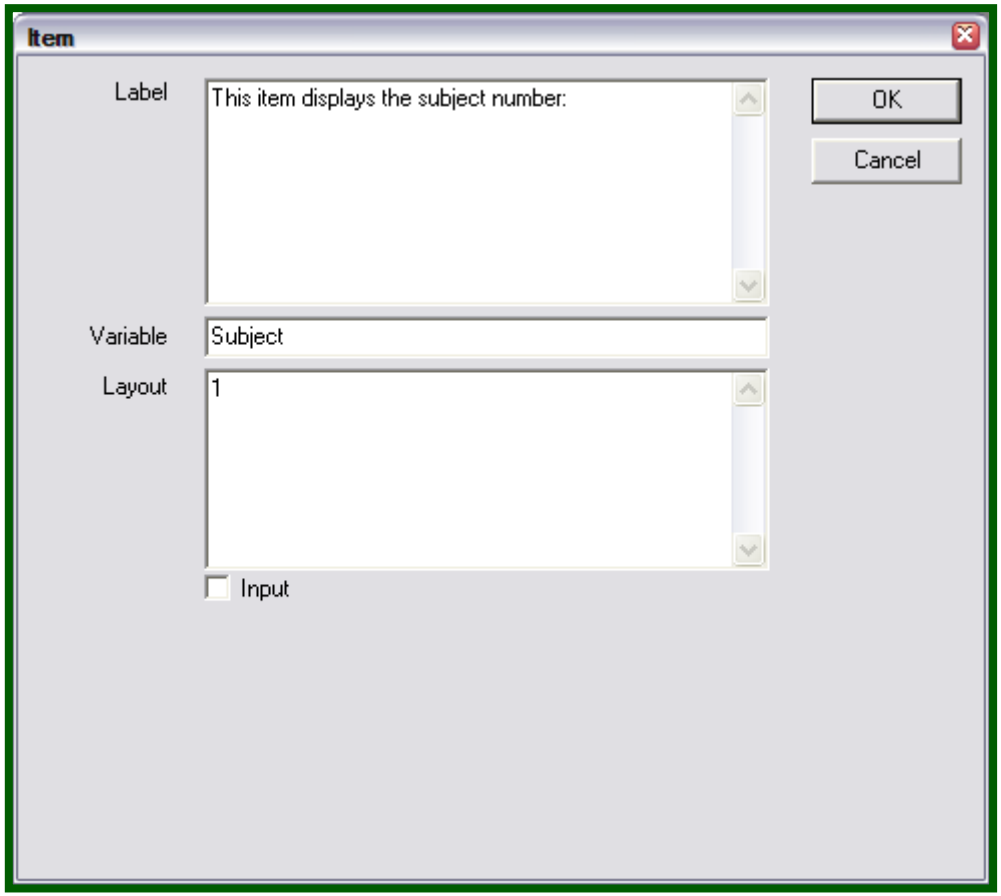
* Inspired by lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

Items: Text output



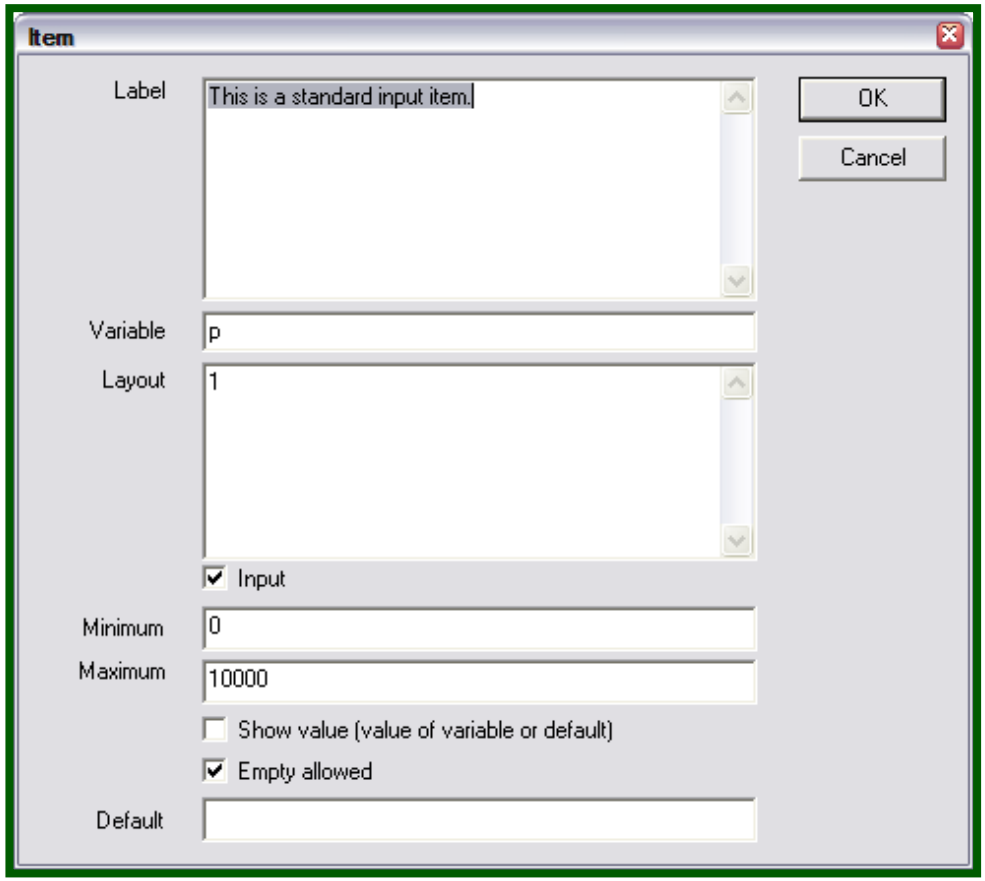
- Result: This is a standard output box.

Items: Variable output



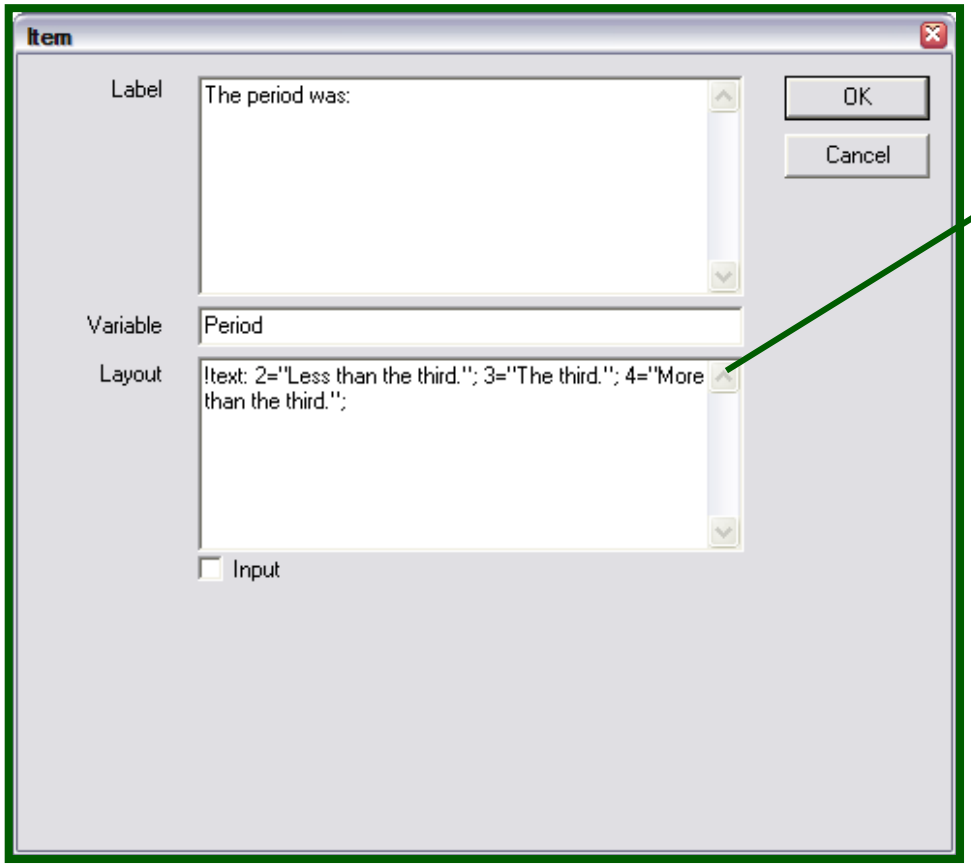
- Result: This item displays the subject number: 1

Items: Variable input



- Result:

Conditional output



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.

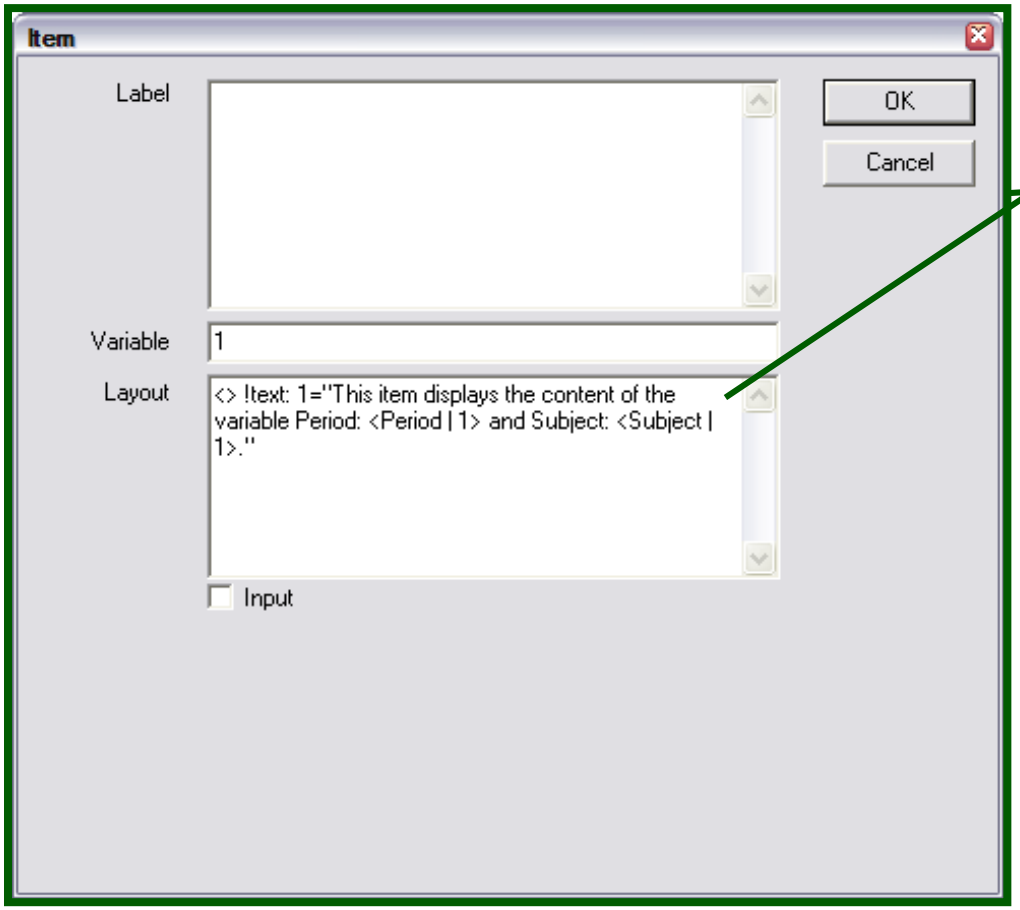
- Results:

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

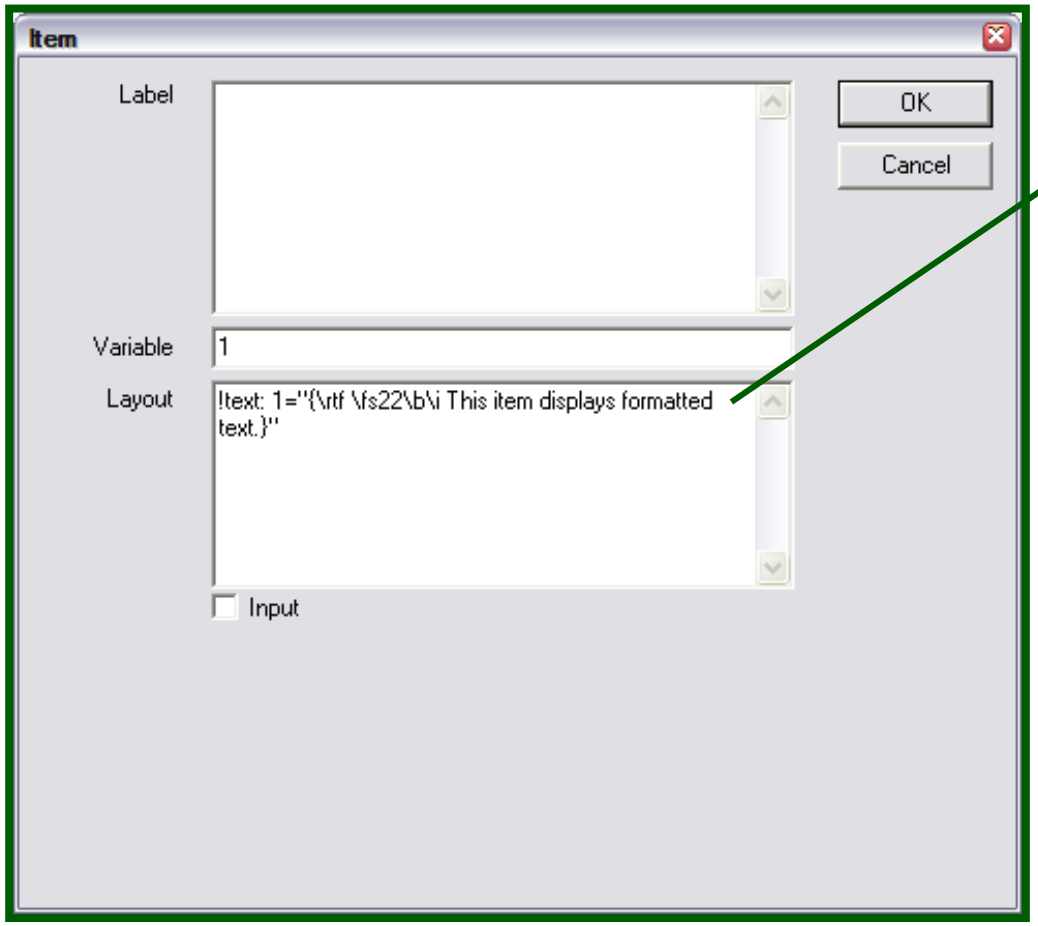


Layout:

```
<> !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

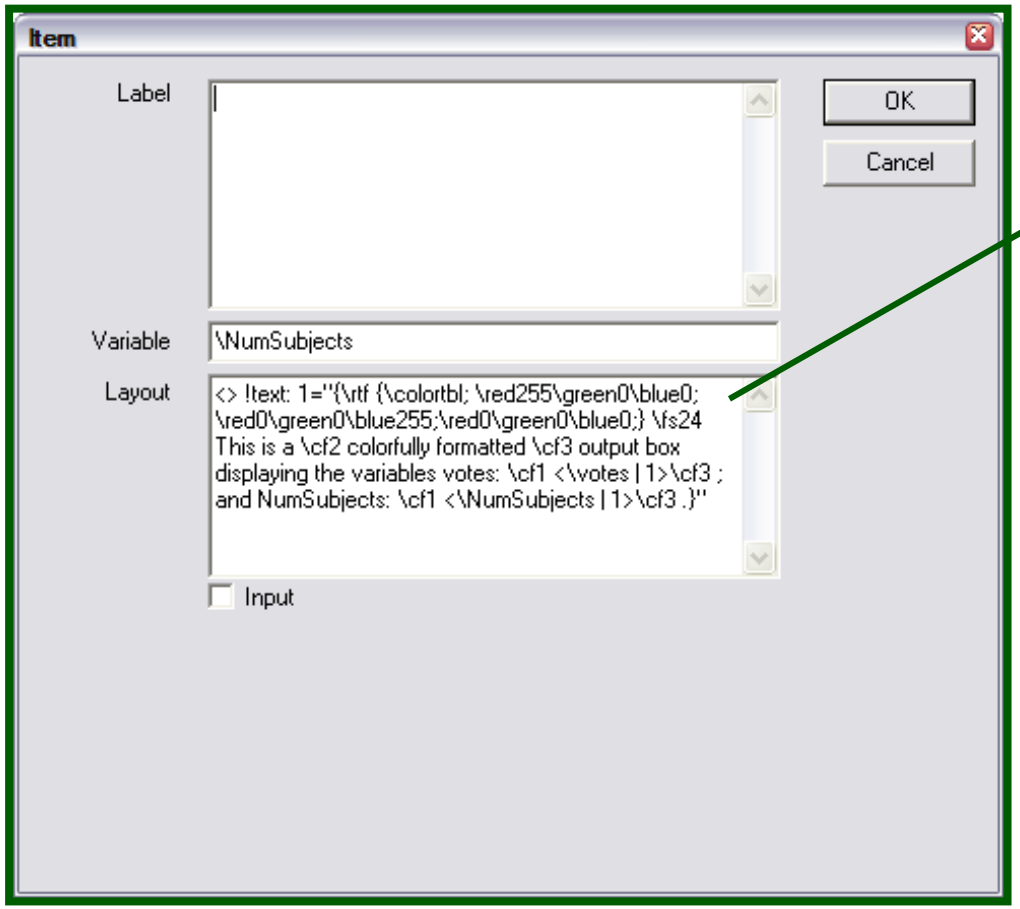


Layout:

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

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

Variables embedded in colored text

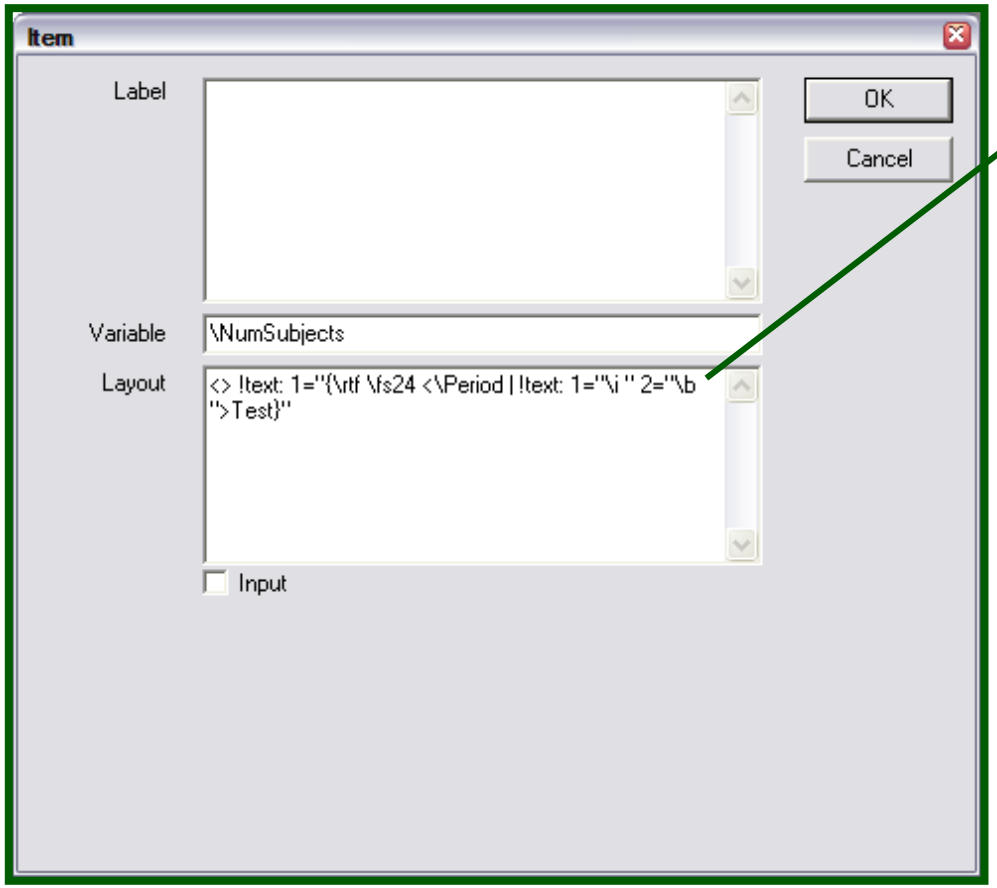


Layout:

```
<> !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



Layout:

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

- Result:

Period 1: *Test* Period 2+: **Test**

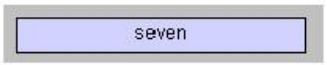
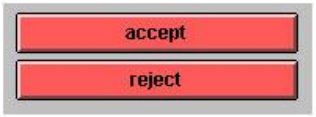
Rich text formatting

Switch	Effect
\tab	tabulator
\par	new paragraph
\line	new line
\bullet	thick dot·
\ql	aligned to left
\qr	aligned to right
\qc	centered
\b	bold
\b0	not bold
\i	italic
\i0	not italic
\sub	small and inferior numbers (index)
\super	small and superior numbers (exponent)
\strike	crossed through
\ul	underline
\ul0	do not underline
\cfn	Text color. n is the index of the color table which is defined by colortbl. See example.
\fsn	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.

Source: Reference manual, p. 56-57.

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";		

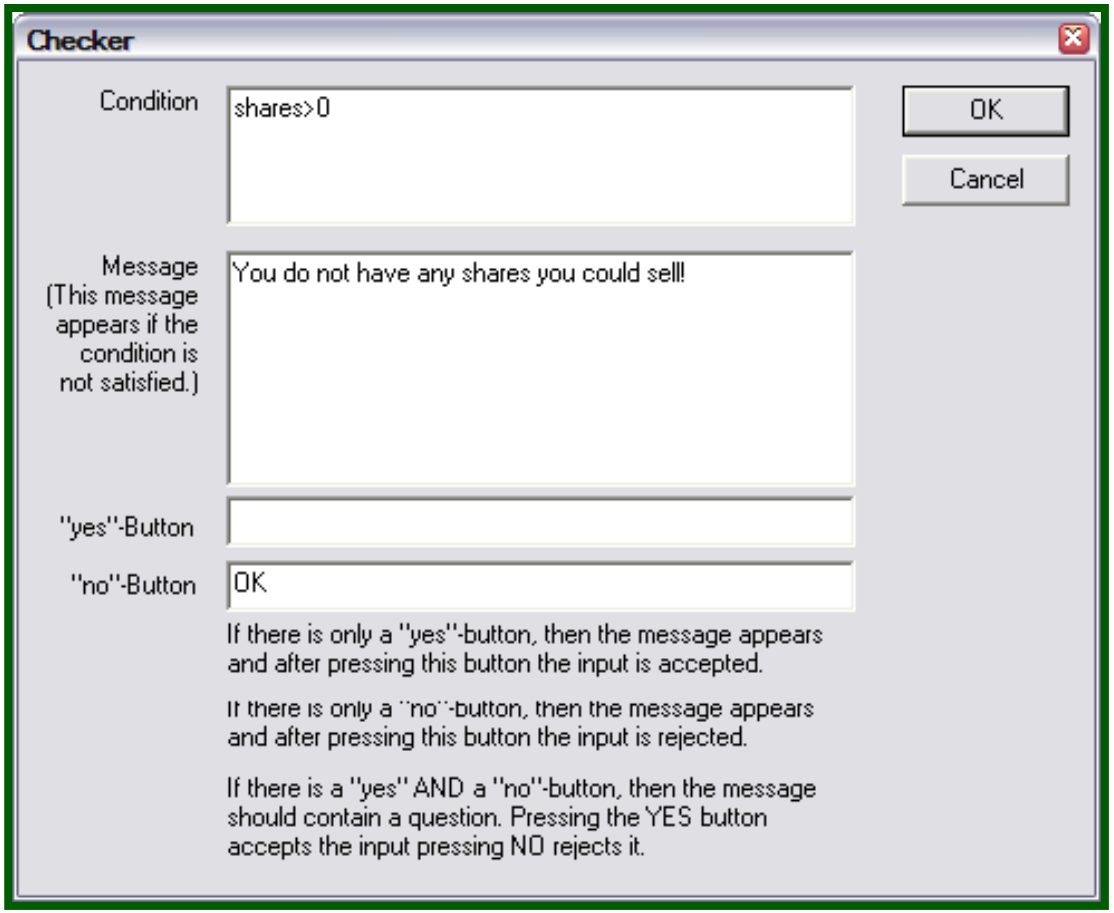
Source: Reference manual, p. 30.

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…

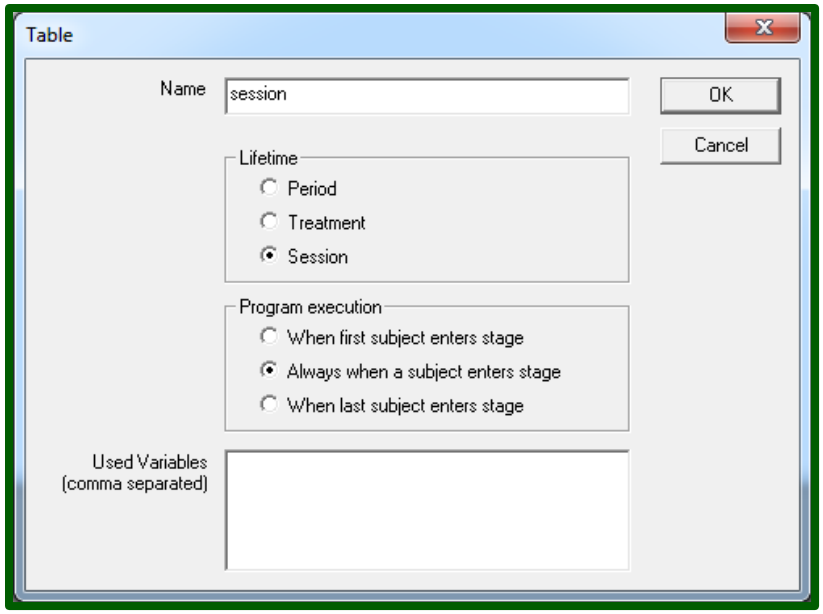


Tables and Scope

- 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



1 subjects	Period	Subject	Group	Profit	TotalProfit	Participate	Money	Stock
1 subjects	1	1	1	0	0	0	6392	10
1 subjects	1	2	1	0	0	0	320	7
1 subjects	1	3	1	0	0	0	6408	3
1 subjects	1	4	1	0	0	0	64	0
2 subjects	Period	Subject	Group	Profit	TotalProfit	Participate	Money	Stock
2 subjects	1	1	1	18.5066667	18.5066667	0	1388	5
2 subjects	1	2	1	1.26666667	1.26666667	0	95	5
2 subjects	1	3	1	25.96	25.96	0	1947	0
2 subjects	1	4	1	2.30666667	2.30666667	0	173	0
2 subjects	Period	Subject	Group	Profit	TotalProfit	Participate	Money	Stock
2 subjects	2	1	1	6.69333333	25.2	0	1890	4
2 subjects	2	2	1	3.21333333	4.48	0	336	4

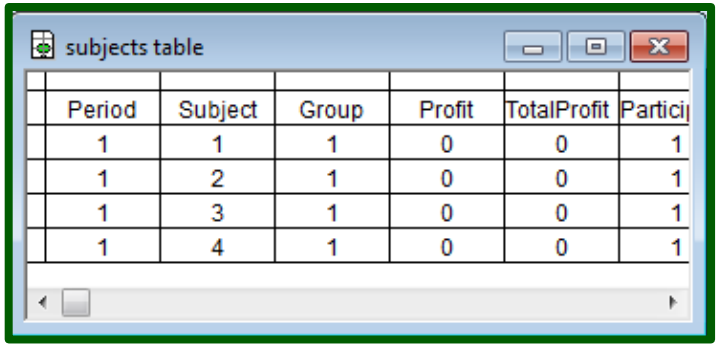


Table subjects

- 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

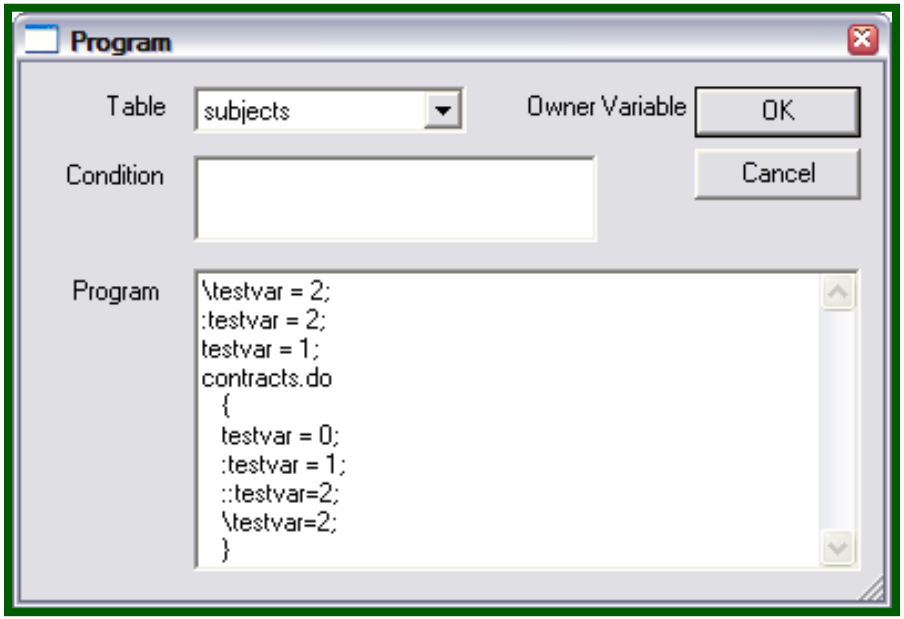
1	contracts	Period	Seller	Buyer	Group	p	Traded	contractID	tradeID	Maker
1	contracts	1	10	8	1	20	1	1	1	10
1	contracts	1	5	8	1	40	1	2	11	5
1	contracts	1	4	-2	1	120	0	3	0	4
1	contracts	1	-2	8	1	100	0	4	0	8
1	contracts	1	6	8	1	30	1	5	2	6
1	contracts	1	4	6	1	40	1	6	6	6
1	contracts	1	2	8	1	40	1	7	3	2
1	contracts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



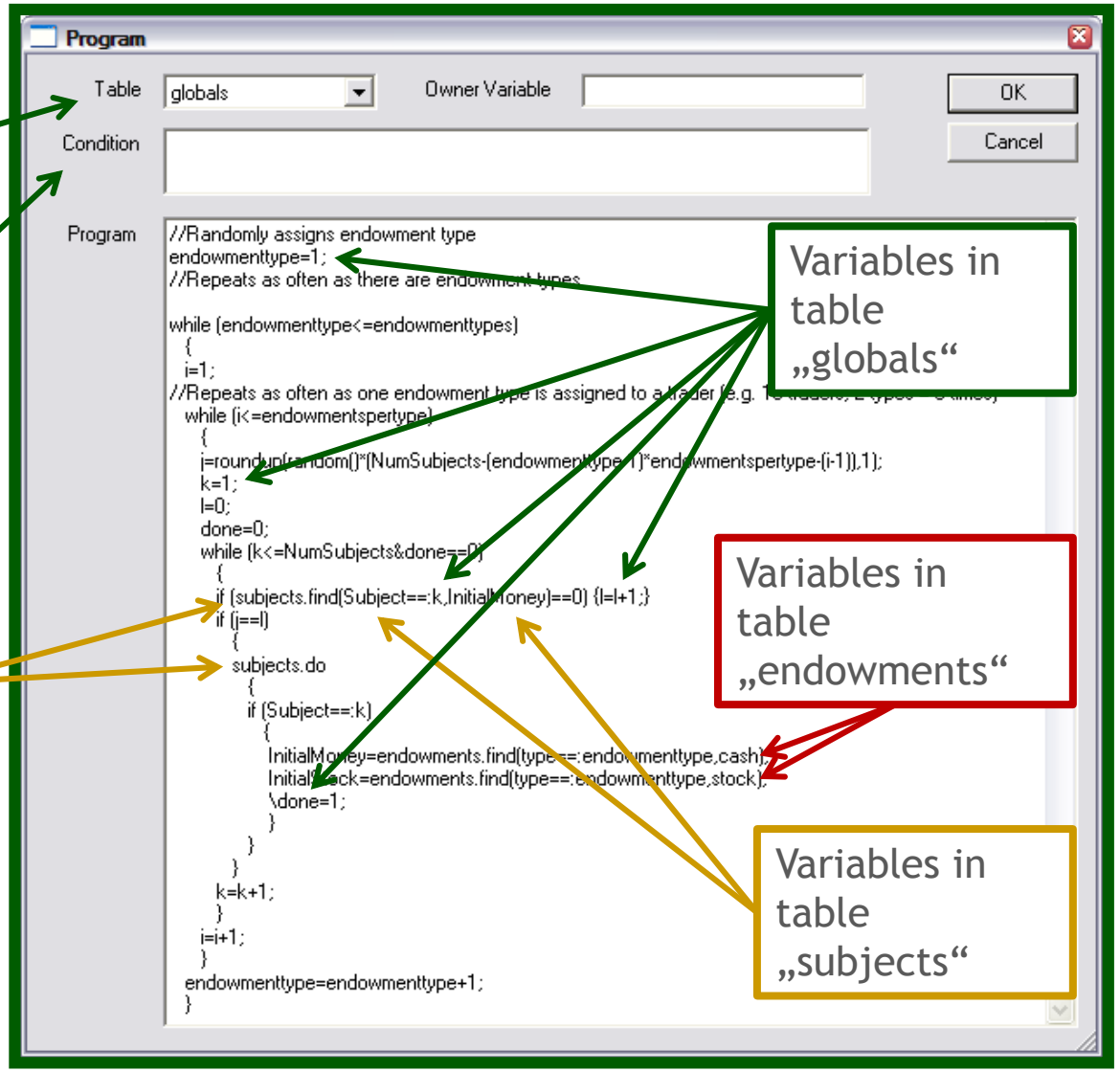
Scope operators:

: one step up

\ all steps up to „globals“ table

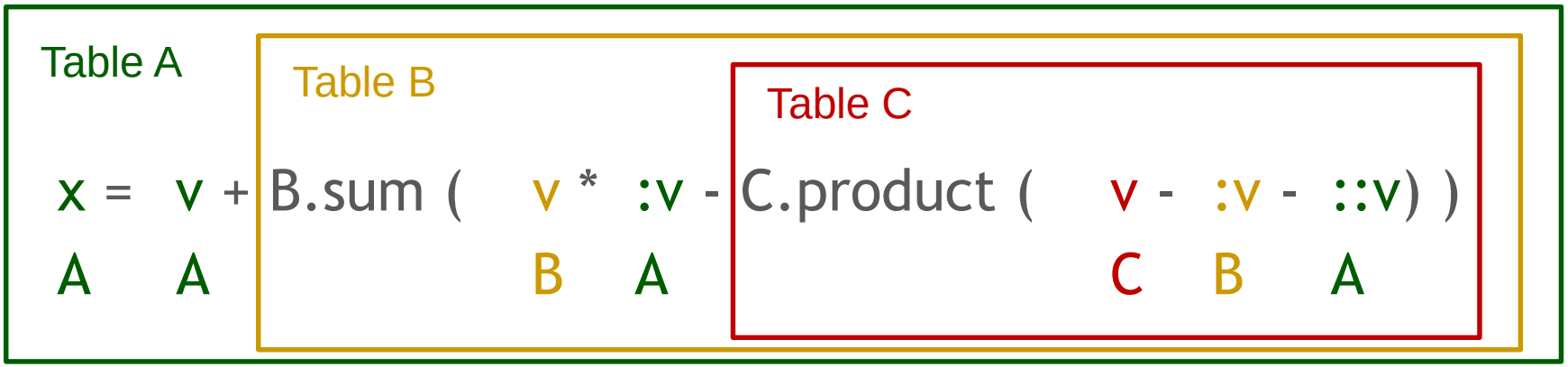
Programs, tables and scope operators

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



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.

Source of presentation form: Esarey, J., “zTree Workshop: Fundamentals of zTree”, http://userwww.service.emory.edu/~jesarey/ ztree.ppt, 2005, p. 81.

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

Exercise: Dividends

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

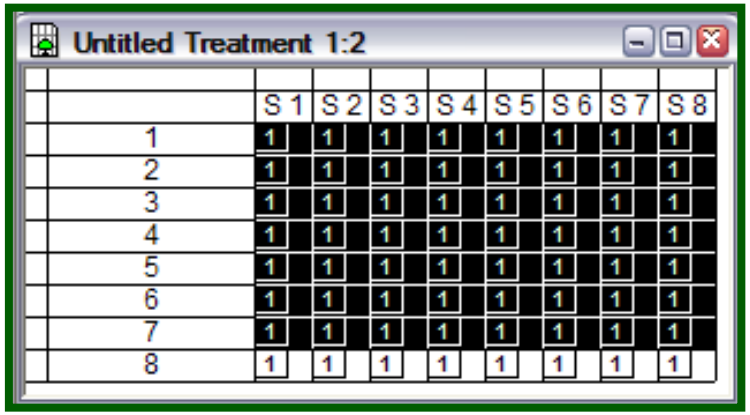
- Built-in matching capabilities
- User-generated matching

Matching schemes

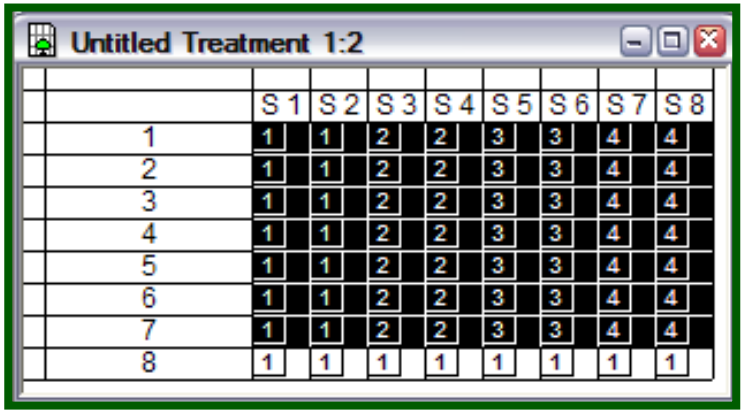
- Matching in z-Tree via group membership
- Group variable pre-defined in subjects table
- Groups are defined in Specific parameters in Parameter table
- Groups can be modified using Group variable in subjects table
- Remember period structure

Group matching

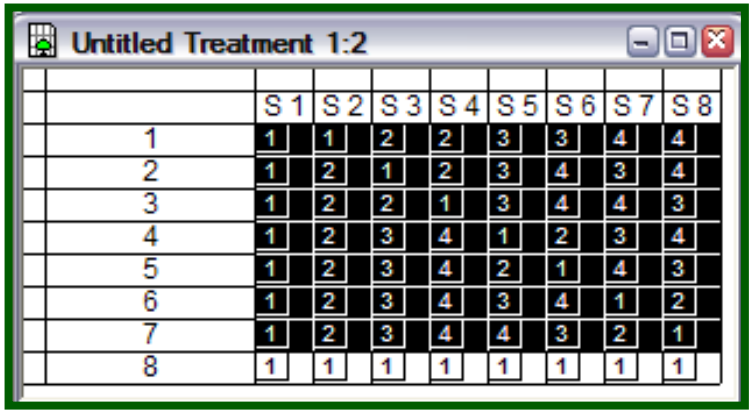
- Select desired periods in parameter table:



- Choose matching procedure from Treatment menu:
  - Partner:

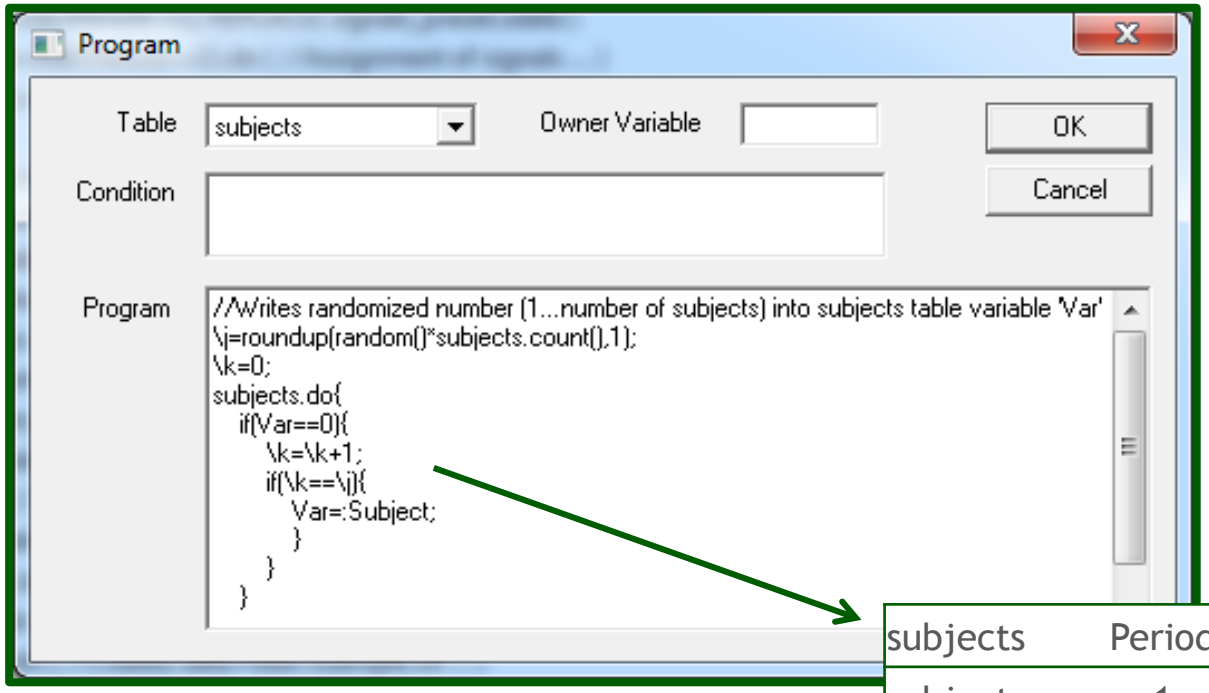


  - Absolute stranger:



Randomly ordering table entries

- Useful for random assignment of groups or endowment types



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

Modulo function

- Usage: mod(x,y)
- Returns the remainder of division of x by y
- Example: Group assignment
  - 3 groups
  - Based on variable SortVar
  - Command:
    Group=
    mod(SortVar-1,3)+1

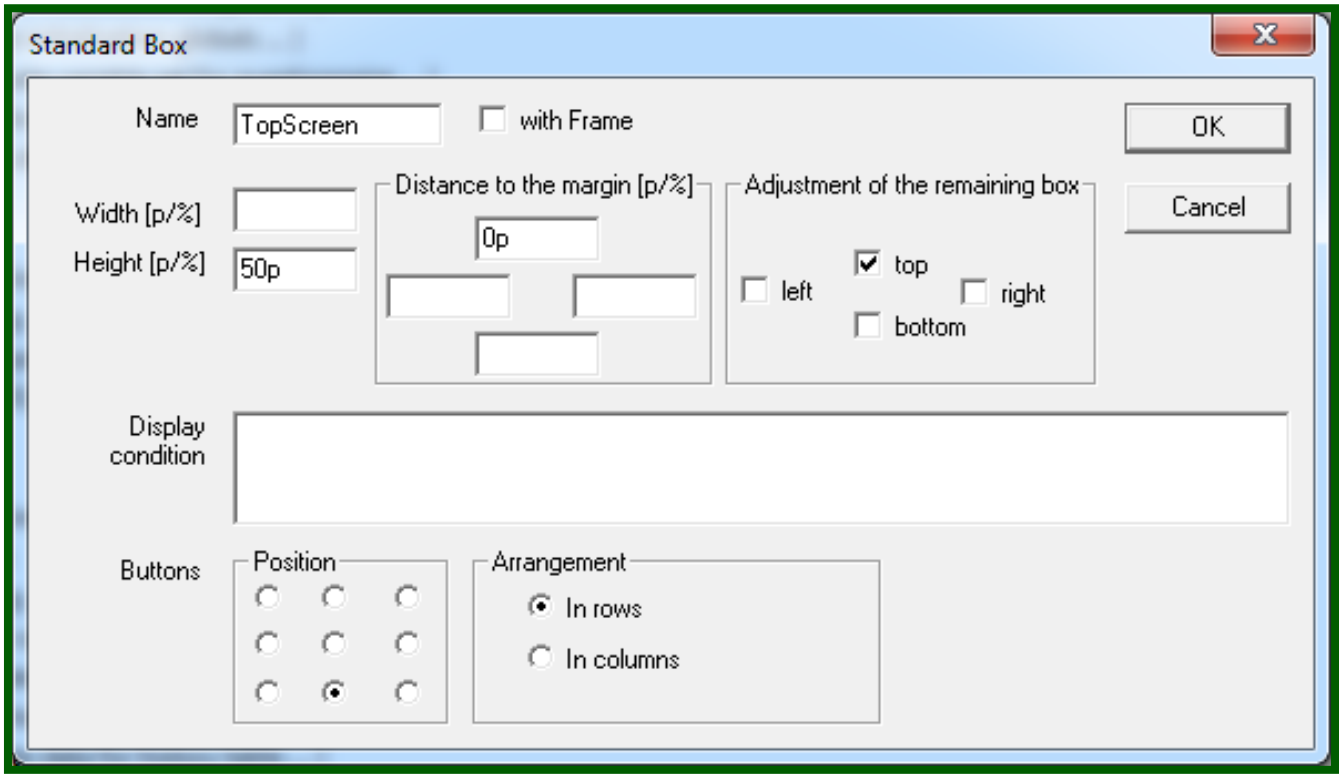
SortVar	mod(SortVar-1,3)	Group
1	0	1
2	1	2
3	2	3
4	0	1
5	1	2
6	2	3

Box Types

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

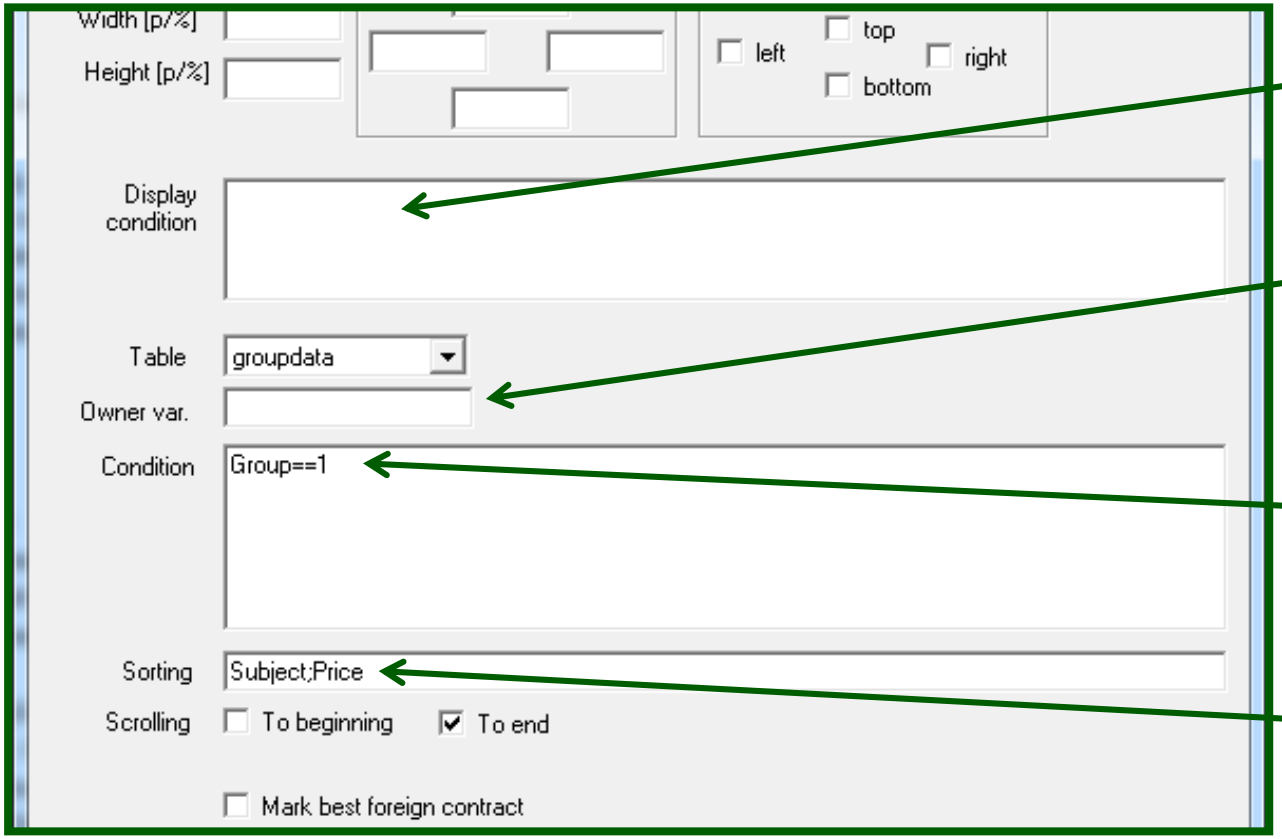
Container box:

- Used to organize other boxes on the screen



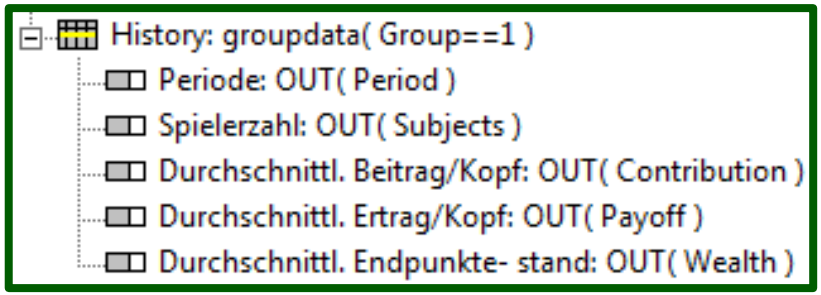
Contract list box:

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



Contract list box:

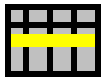
- Code:



- 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

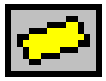
Box types

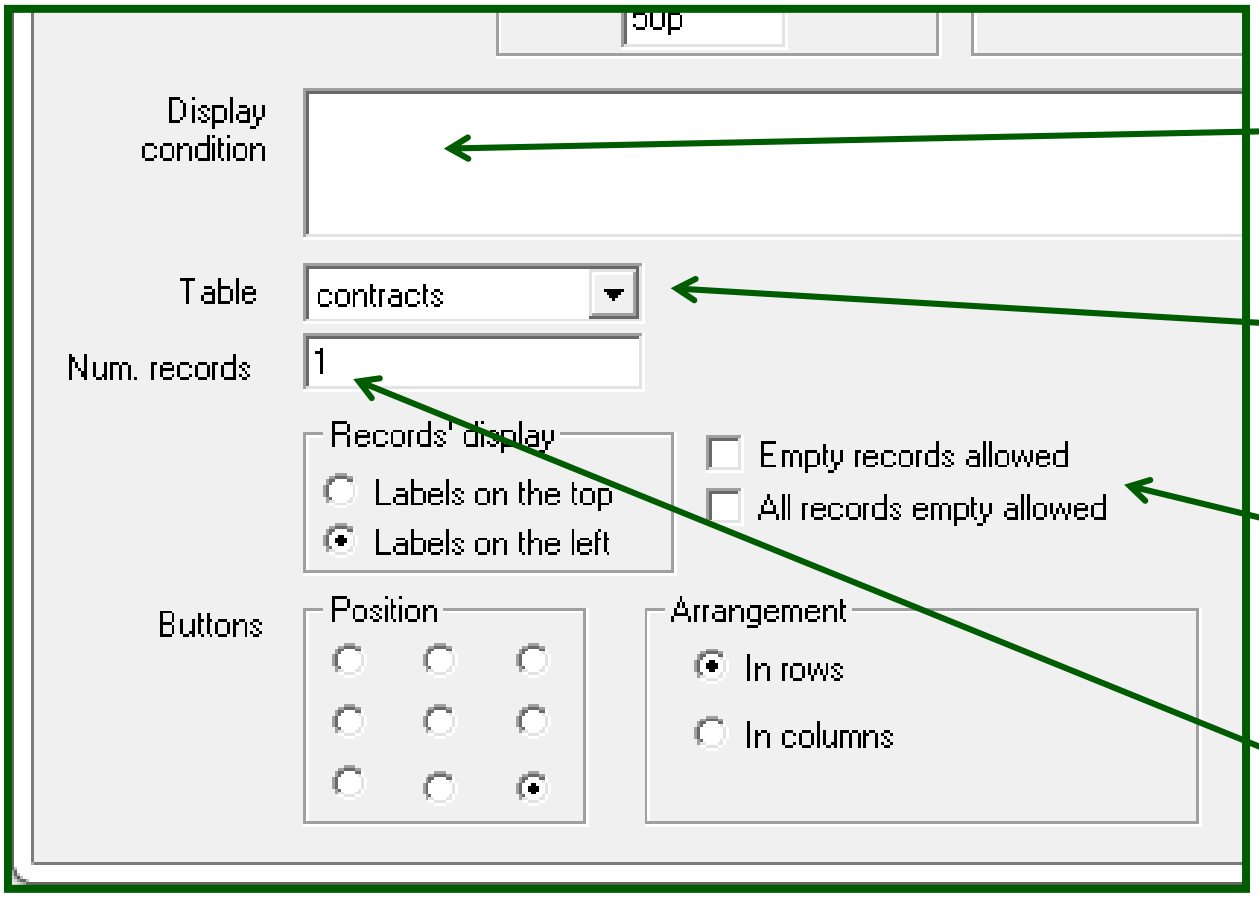
Contract list box: 

- 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:

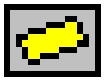


Condition for displaying box

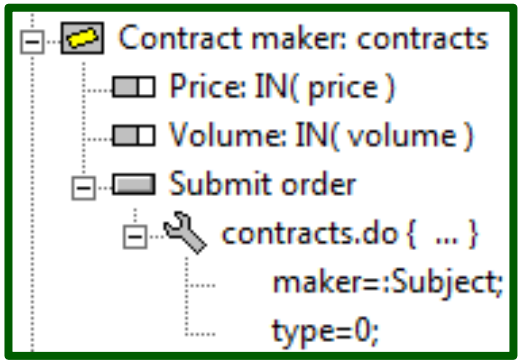
Table wherein new records should be created

Additional settings for contract creation

Number of contracts which can be entered simultaneously

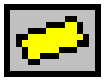
Contract creation box: 

- Code:



- Output:

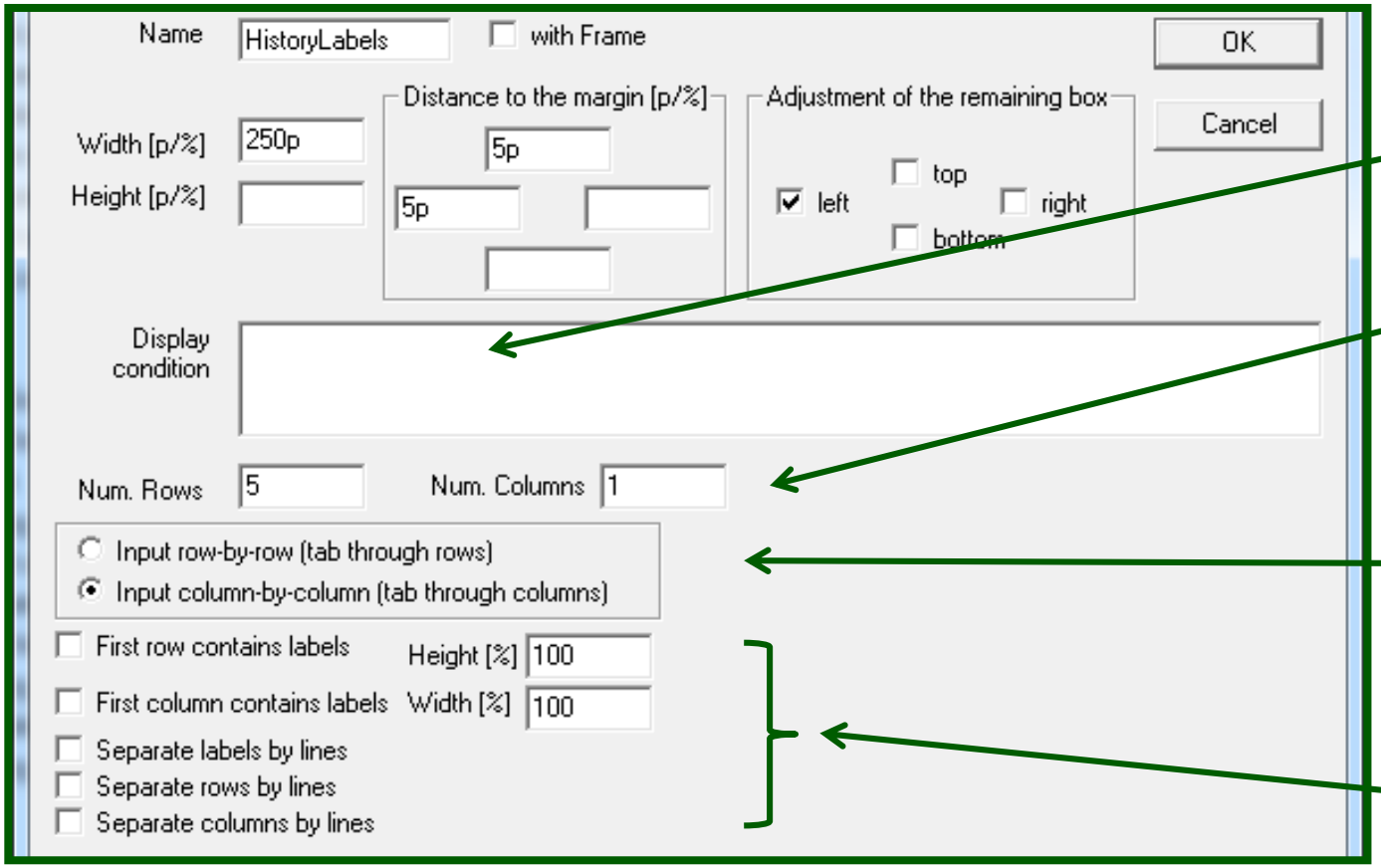
Box types

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

Grid box:

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



Condition for displaying box

Defines size of the box

Determines how the ordering of the items is interpreted

Formatting options

Grid box:

- Code:

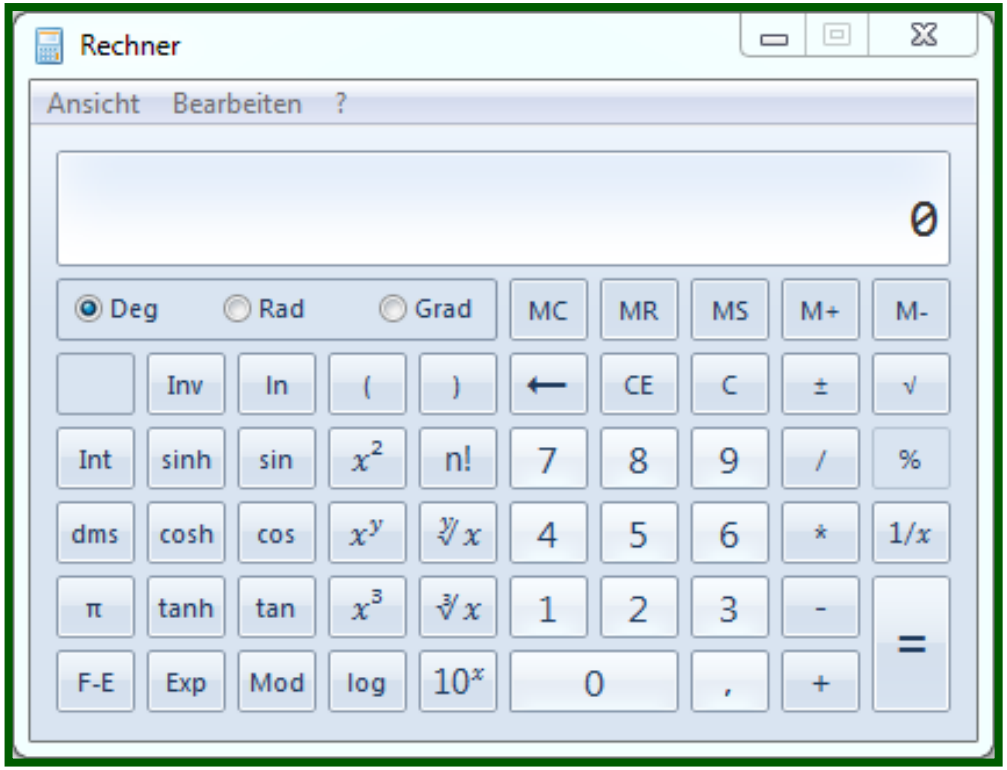
Attention:
Drawing a grid box with many items takes z-Tree a long time. Whenever possible, use e.g. a contract list box instead.

- 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						

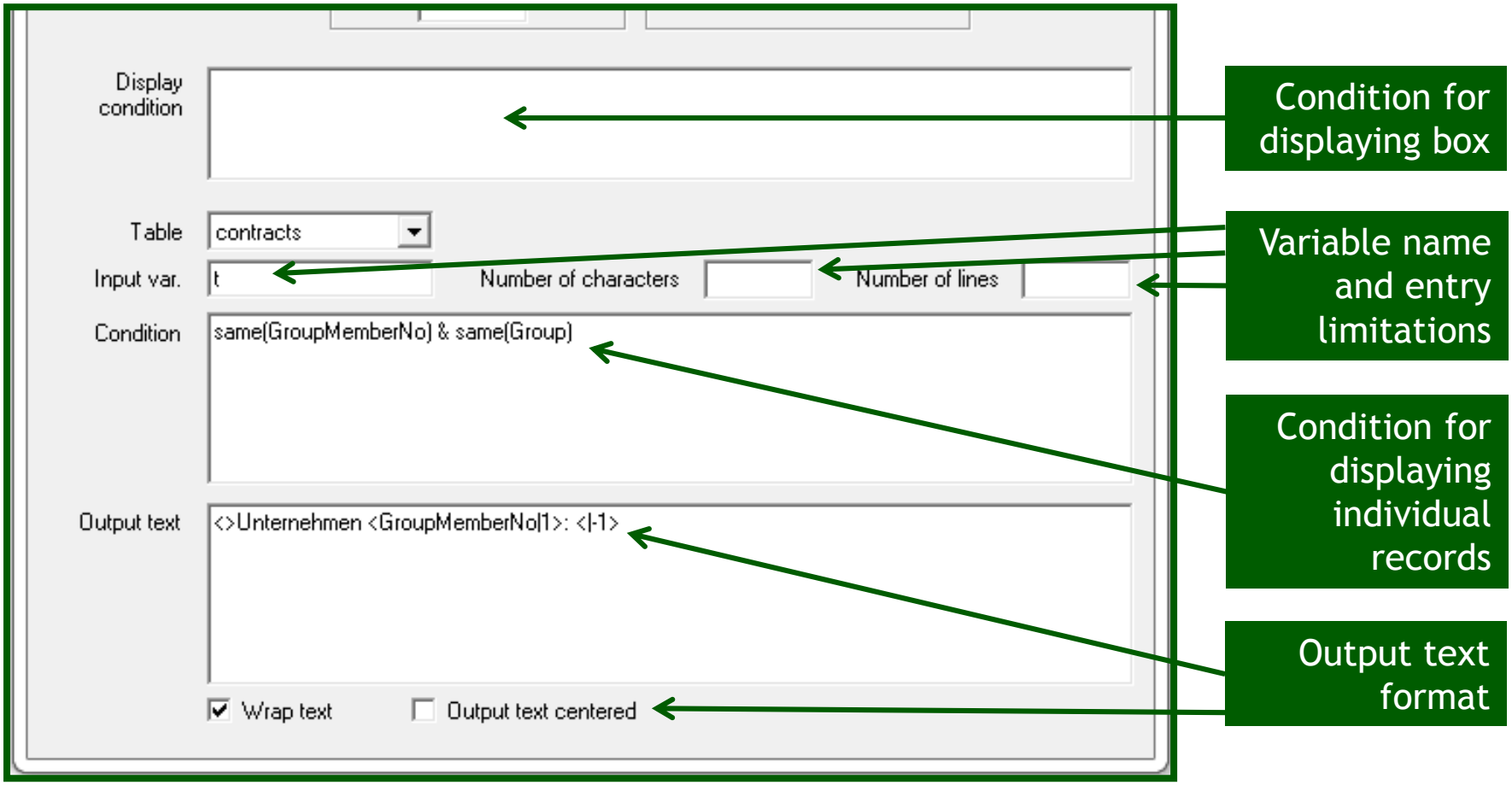
Calculator buttonbox:

- Gives subjects access to the Windows calculator



Chat box: 

- Used to display/exchange chat messages:

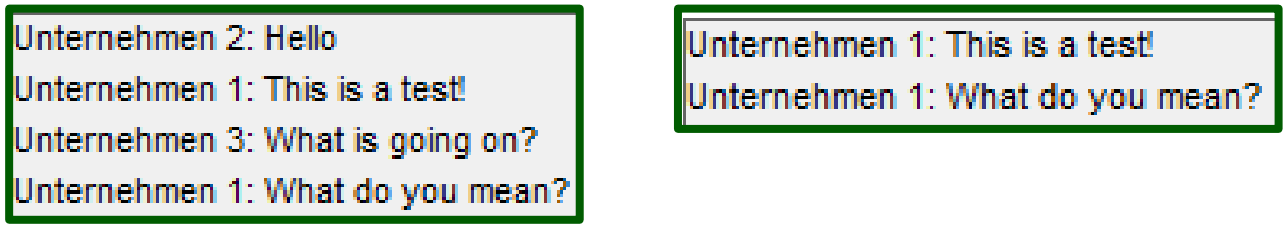


Chat box:

- 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:



Subject Progression

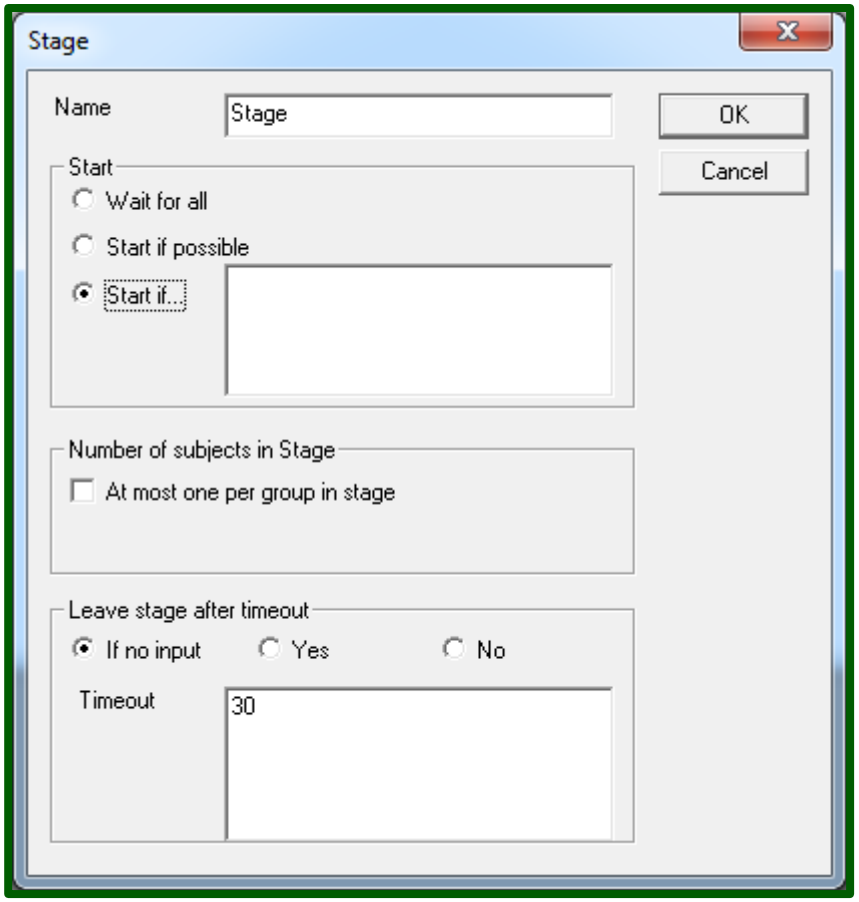
- Periods and stages in z-Tree
- Stage options
- Controlling subject progression
  - Subjects in different stages
  - Subjects in different “periods”
  - Random ending period

General notes

- 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



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

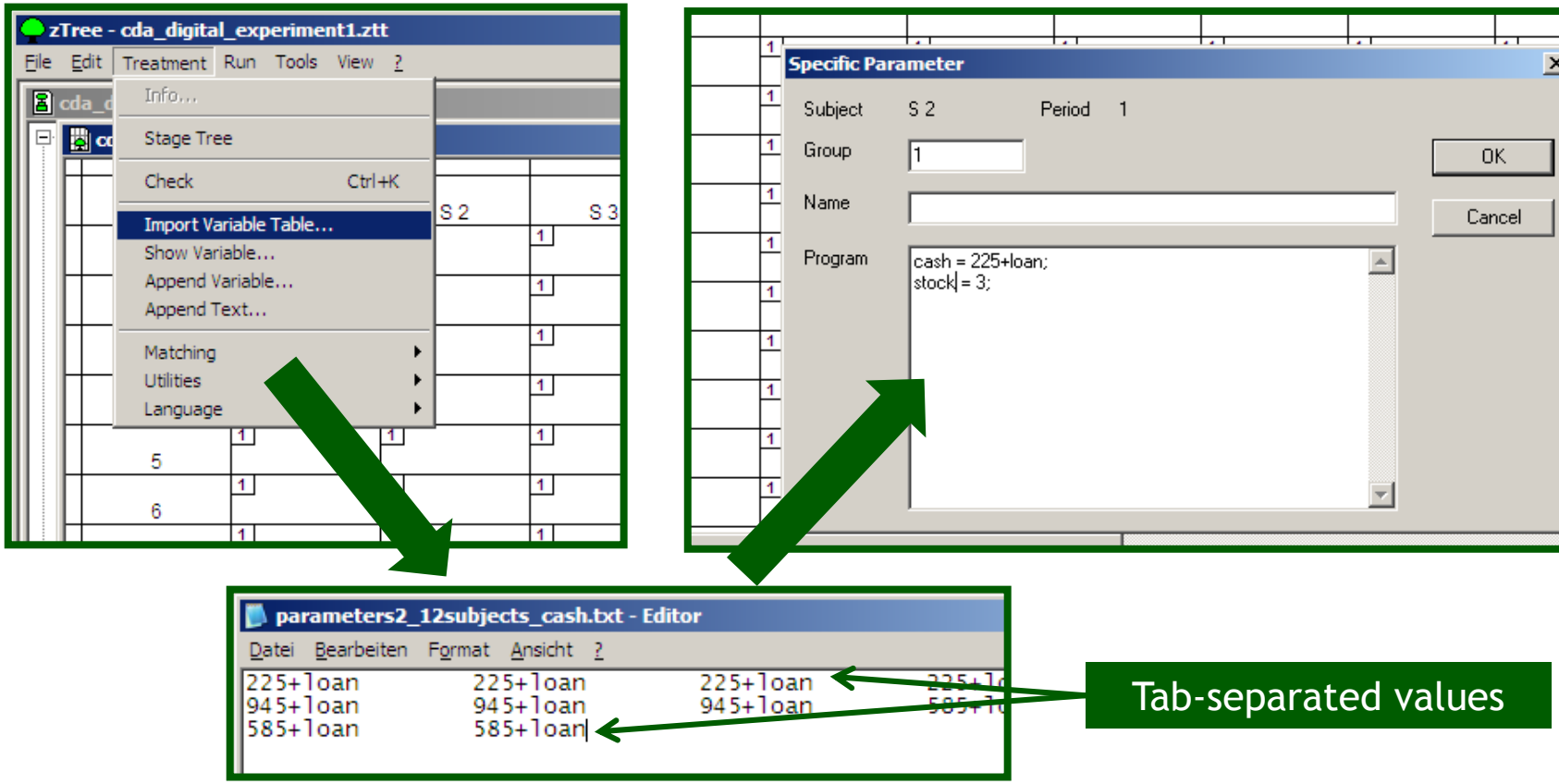
- Program everything in a single stage, using stacked containers/boxes
- Switch between different screens using display conditions
- Use user-generated period variable (e.g. SubjectivePeriod)
- Use user-generated header
- Put programs into buttons instead of stage backgrounds

Import and Export

- Parameter table import
- Table dumper and table loader

Parameter table variable import

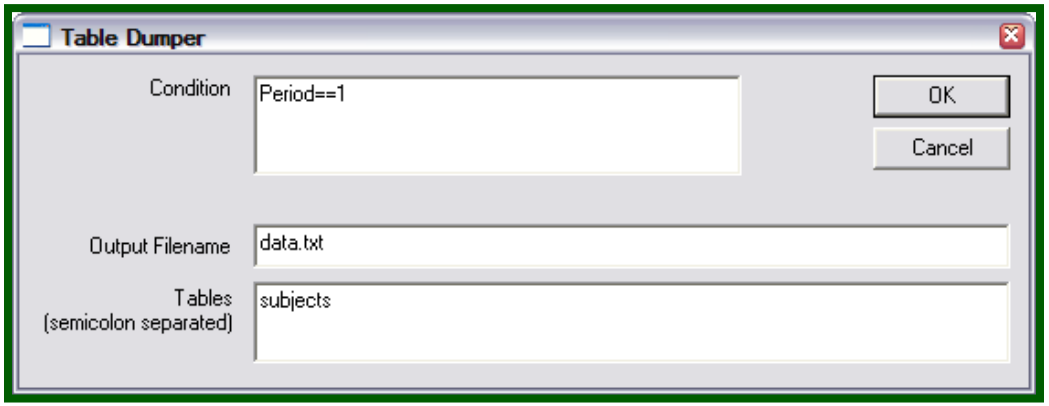
- Variables can be imported from ASCII file (one-by-one)
- Allows parameter customization before session

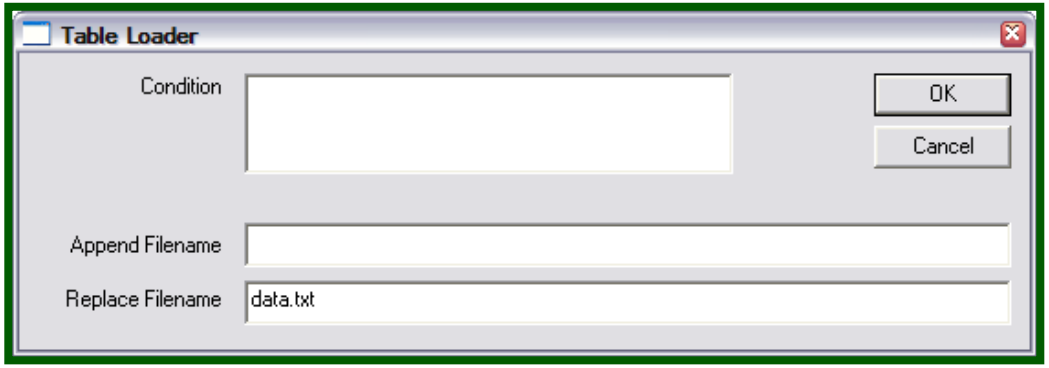


Data import and export 1/3

- 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)





Data import and export 2/3

- 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

Exercise: Data import and export

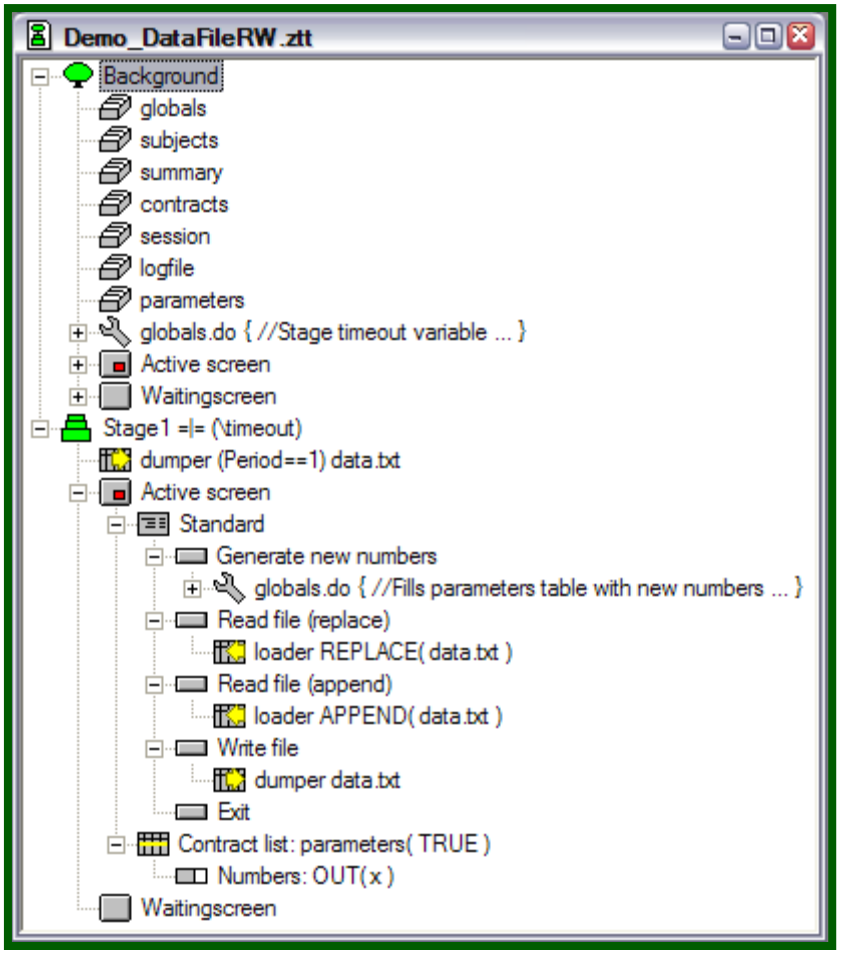
- Program Demo_DataFileRW.ztt

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



Generating repetitive code with Excel

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

Questionnaires

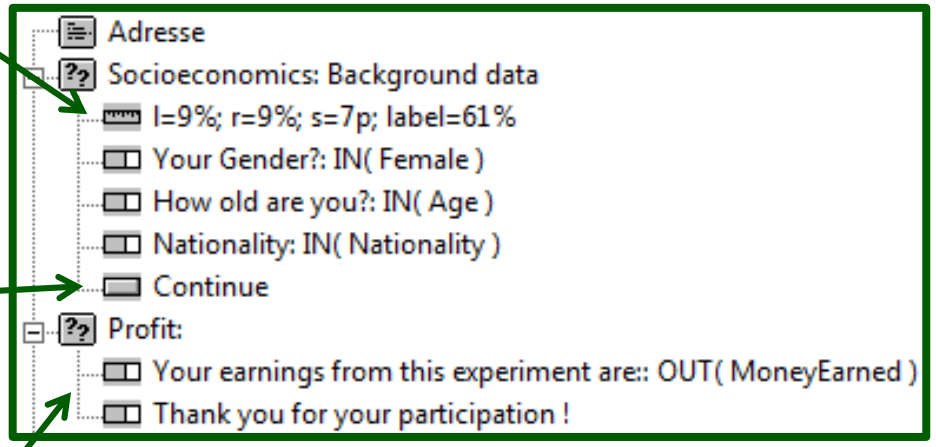
- Overview
- Using the session table
- Checking answers
- Presenting results

Overview

- Run after ≥ 1 treatment (sets number of subjects)
- Adress form [sic!] needed to write payment file (delete components by deleting caption in adress 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



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.

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:
  - <> Displays x: < x | 1 >

Checking answers

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

Presenting questionnaire results

Link to
Presenting questionnaire results
in
Managing z-Tree Output

Managing z-Tree Output

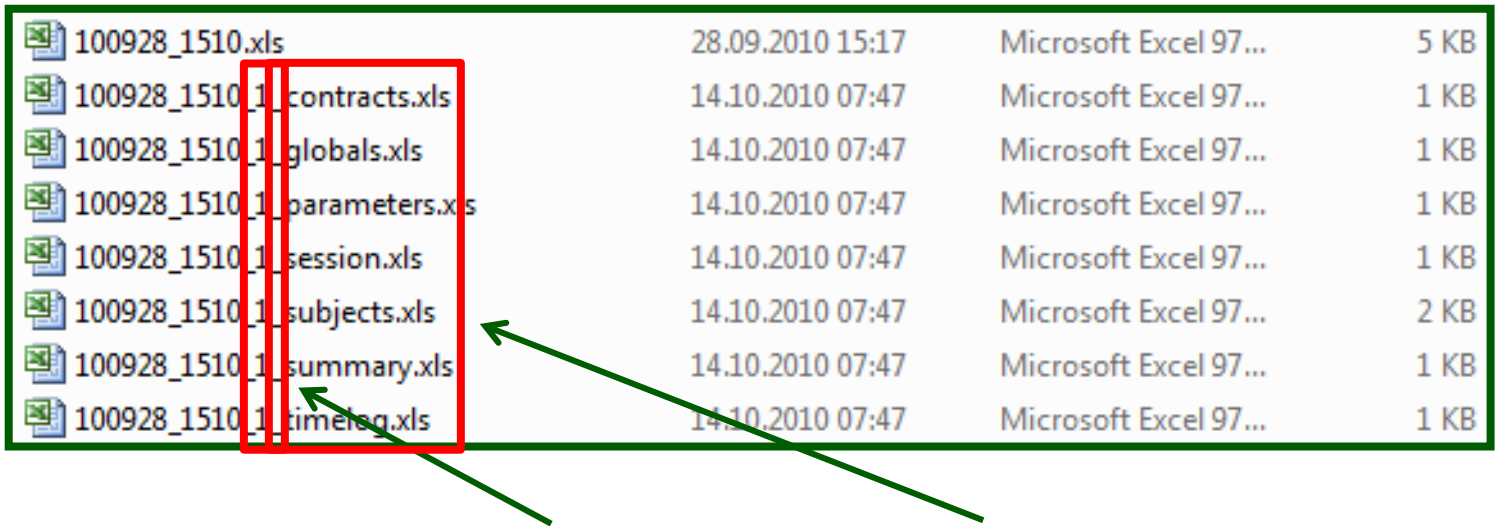
- Reformatting z-Tree output
- Presenting questionnaire results
- Import into R
- Import into and processing in Stata
- Results table

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

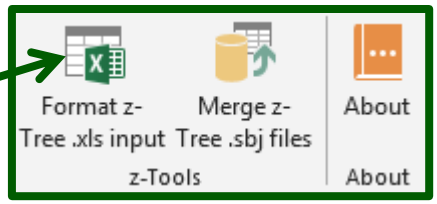
- z-Tree's "Tools - Separate tables…" command
  - Select .xls-file to process
  - Output:



  - Separated by treatments and tables

Reformatting z-Tree output

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



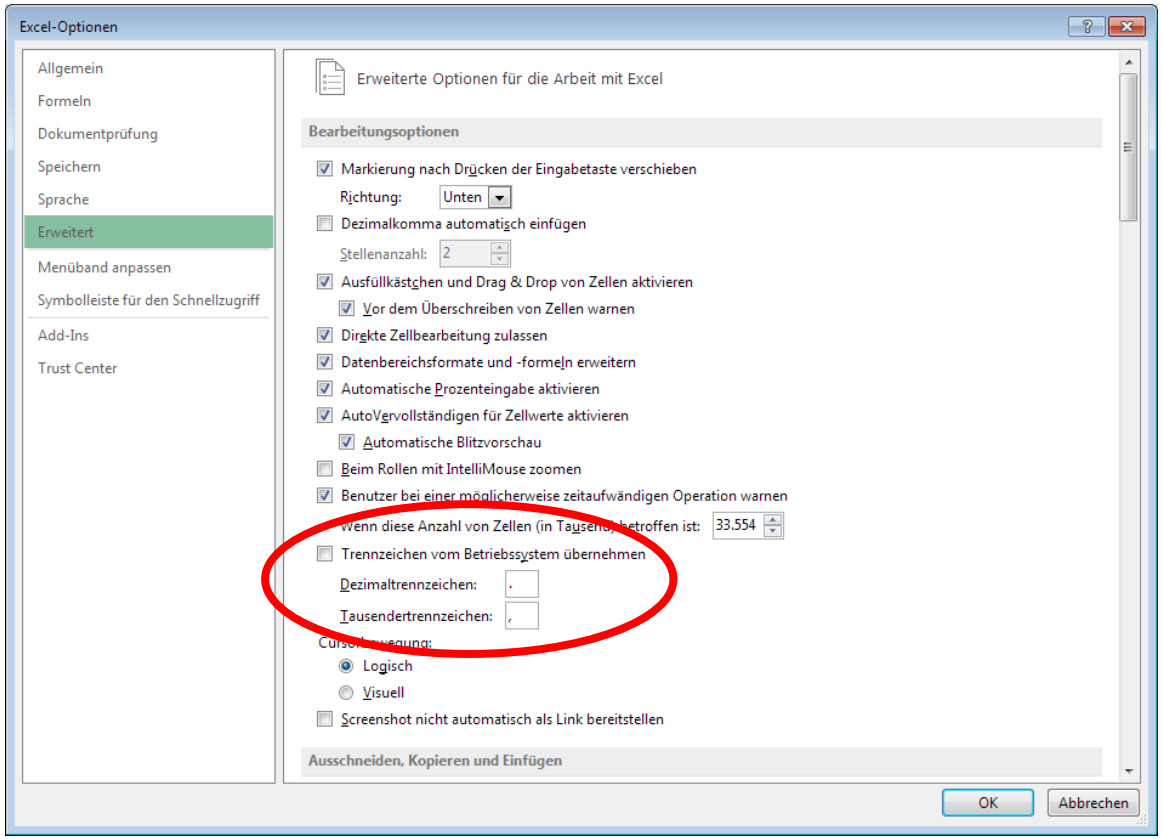
	A	B	C	D	E	F	G	H
1	100304_1001	1	subjects	Period	Subject	Group	Profit	TotalProf
2	100304_1001	2	subjects	1	1	1	188.727273	188.7272
3	100304_1001	2	subjects	1	2	1	171.818182	171.8181
4	100304_1001	2	subjects	1	3	1	13.4545455	13.45454
5	100304_1001	2	subjects	1	4	1	231.818182	231.8181
6	100304_1001	2	subjects	1	5	1	85.7272727	85.72727
7	100304_1001	2	subjects	1	6	1	62.7272727	62.72727
8	100304_1001	2	subjects	1	7	1	81.9090909	81.90909
9	100304_1001	2	subjects	1	8	1	69.0909091	69.09090
10	100304_1001	2	subjects	1	9	1	82.3636364	82.36363
11	100304_1001	2	subjects	1	10	1	105.090909	105.0909
12	100304_1001	2	subjects	1	11	1	0	
13	100304_1001	2	subjects	2	1	1	1.81818182	190.5454
14	100304_1001	2	subjects	2	2	1	48.0909091	219.9090
15	100304_1001	2	subjects	2	3	1	102.727273	116.1818
16	100304_1001	2	subjects	2	4	1	-21.8181818	2

RawData | session | dividends | endowments | globals | subjects

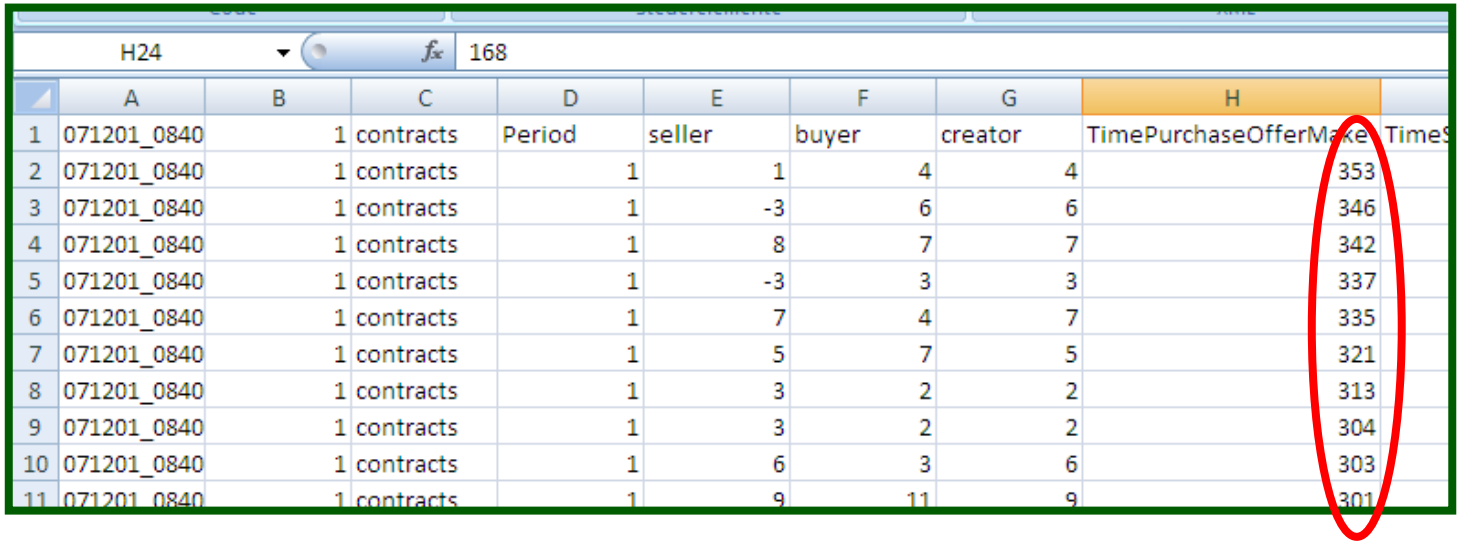
Bereit

Topics in data processing

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



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



H24 | 168

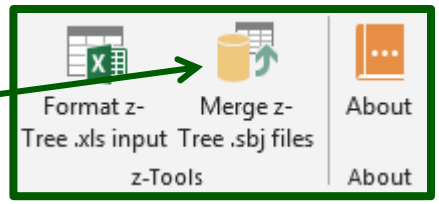
	A	B	C	D	E	F	G	H
1	071201_0840	1	contracts	Period	seller	buyer	creator	TimePurchaseOfferMa
2	071201_0840	1	contracts	1	1	4	4	353
3	071201_0840	1	contracts	1	-3	6	6	346
4	071201_0840	1	contracts	1	8	7	7	342
5	071201_0840	1	contracts	1	-3	3	3	337
6	071201_0840	1	contracts	1	7	4	7	335
7	071201_0840	1	contracts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

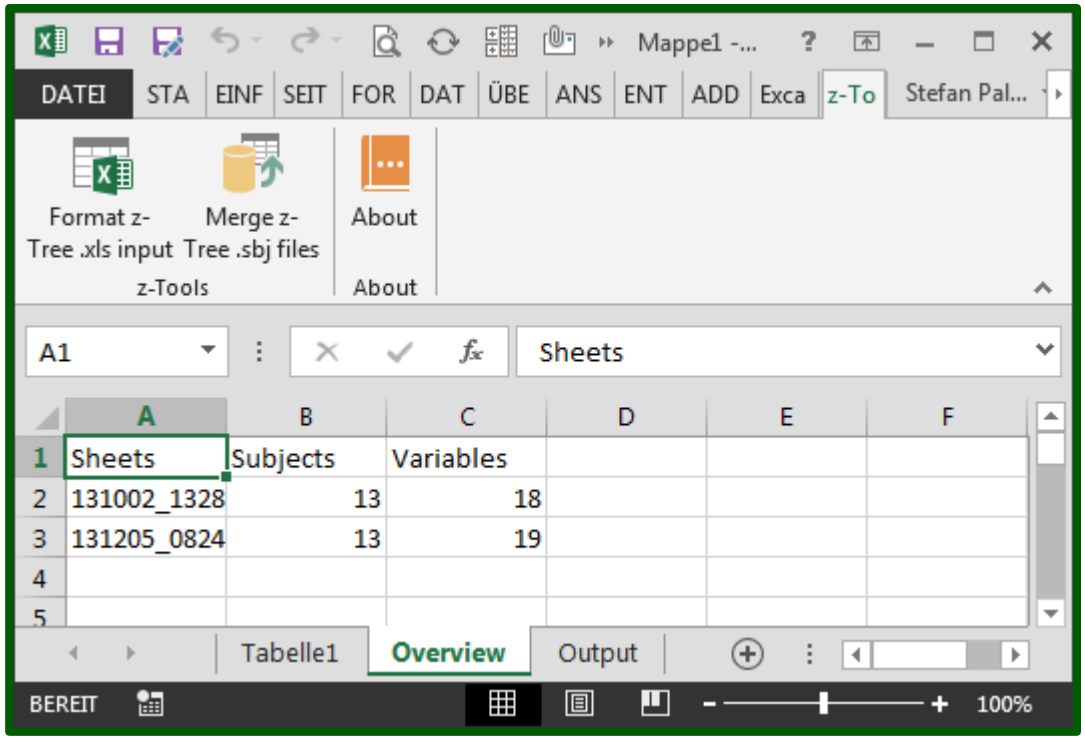
Presenting questionnaire results

- 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

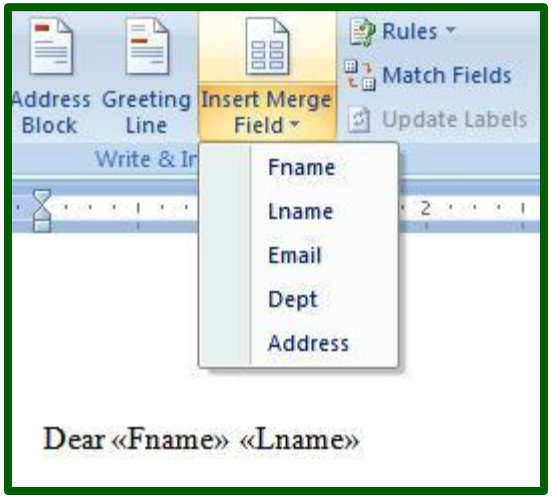
- Using my Excel add-in zTools
  - Click “Merge z-Tree .sbj files”
  - Select questionnaires to merge
  - Output (in same Excel file):

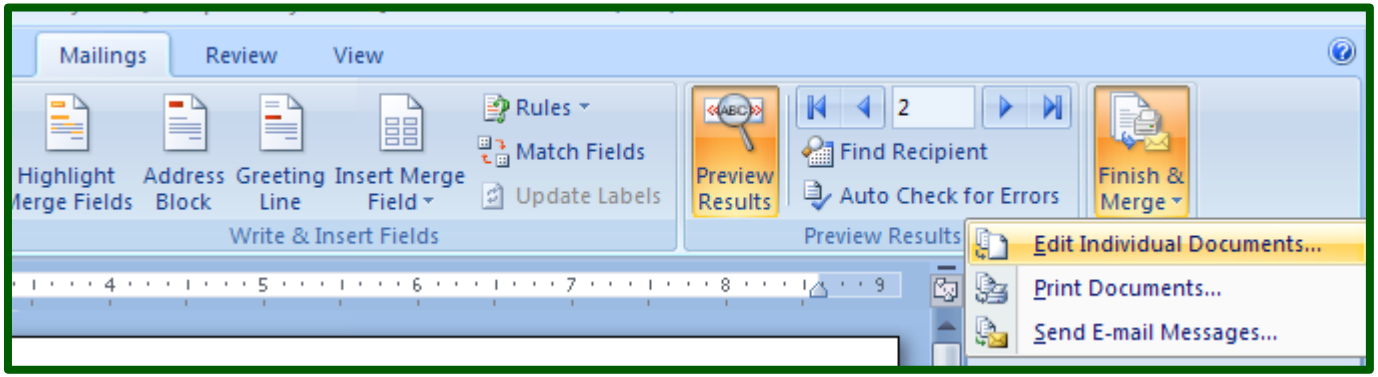




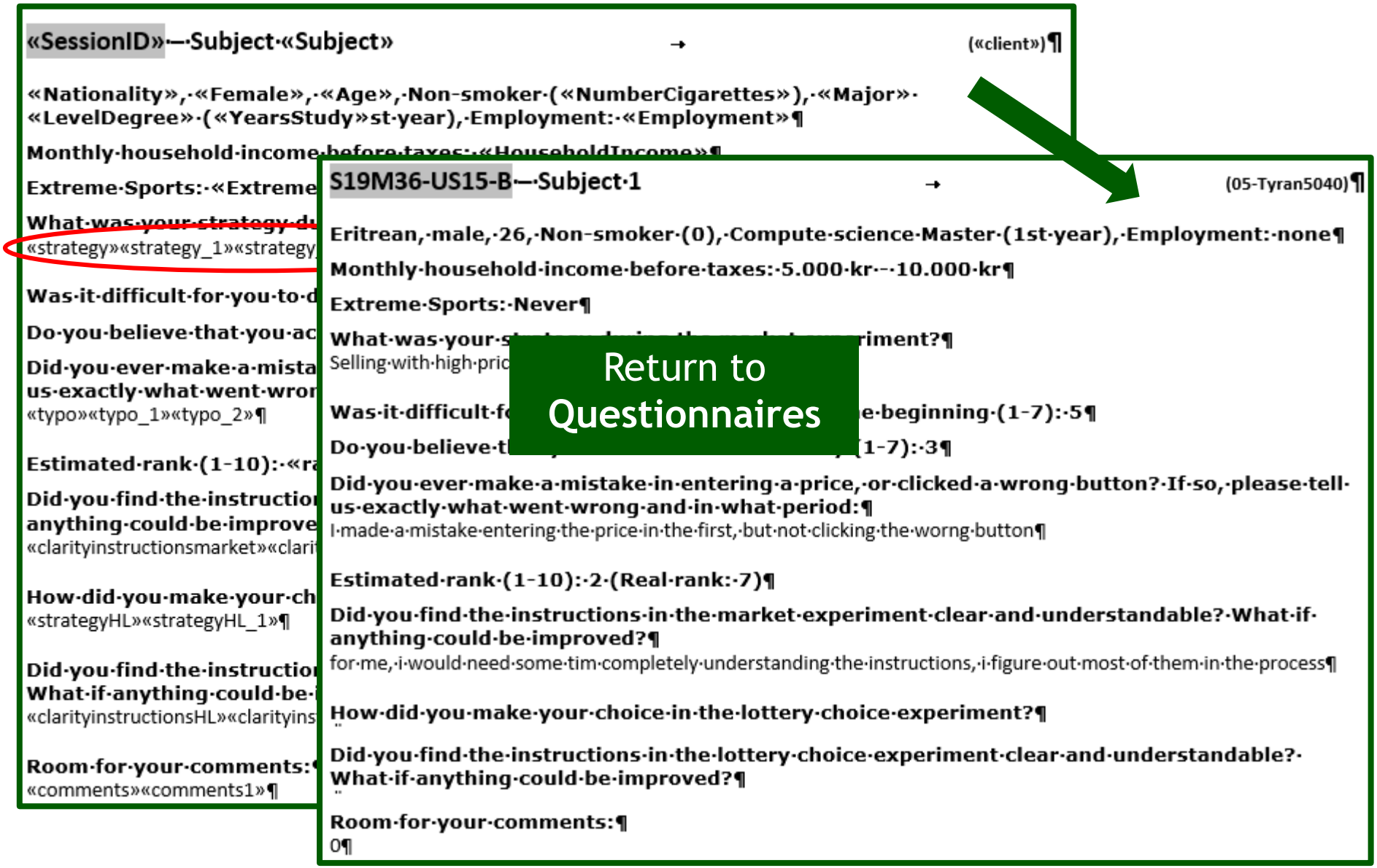
Presenting questionnaire results

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





Presenting questionnaire results



Import into R

- 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))
```

Import into Stata

- 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…

Simple statistics in Stata

- 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)

Results table

- 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

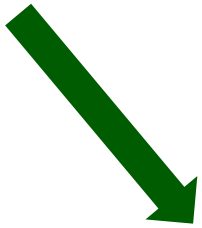
Set number of players=11 (10 participants + 1 experimenter) in treatment and Holt/Laury.

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;
  }
```

Testing

- 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

Documenting

- Keep a documentation of tables and variables:

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.
AccepterID	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.

Debugging*

- 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.

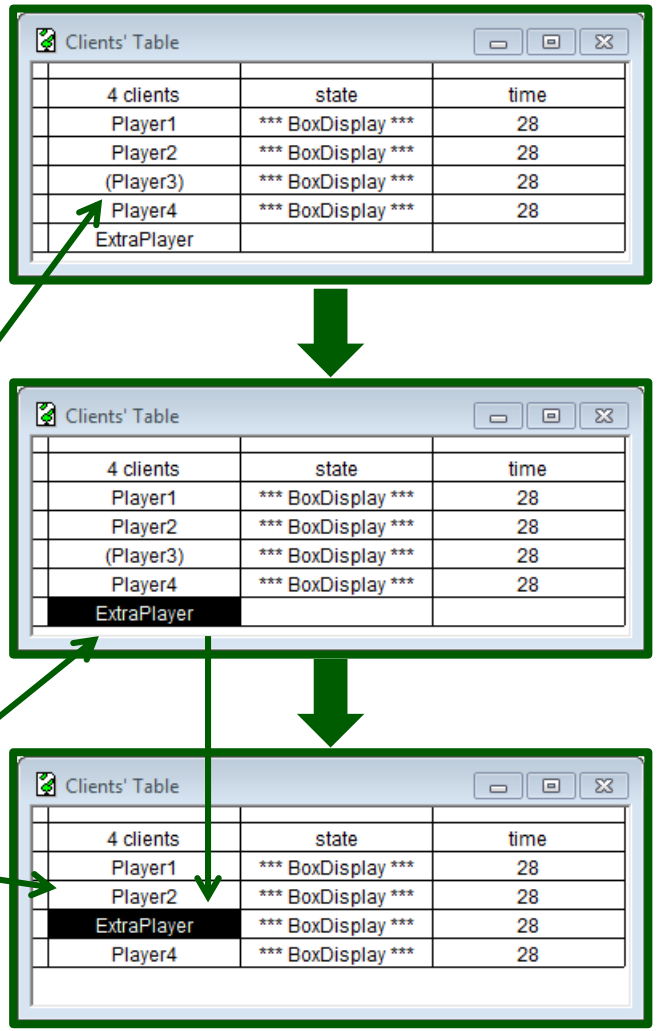
Crash handling*

- 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*

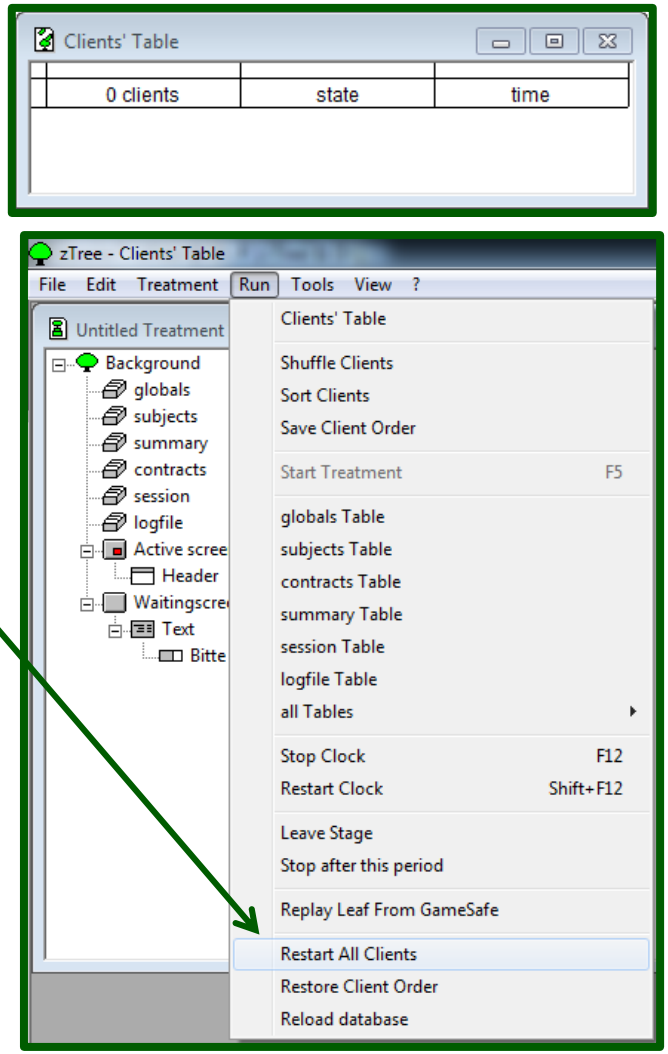
- Client Crash
- If the PC is not working, start a new PC
- Start z-leaf on new PC
- Open clients table on z-tree
- Disconnected client appears in parentheses
- Move new client over field of old client



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

Crash handling*

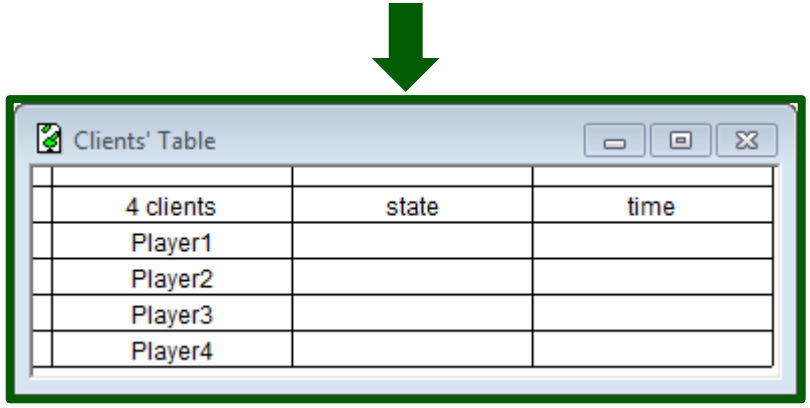
- Server Crash
- Restart z-Tree
- Open the clients' table
- Restart all clients with menu command "Restart all Clients"
- If some clients do not reconnect, manually restart their z-leafs



* 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



* 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

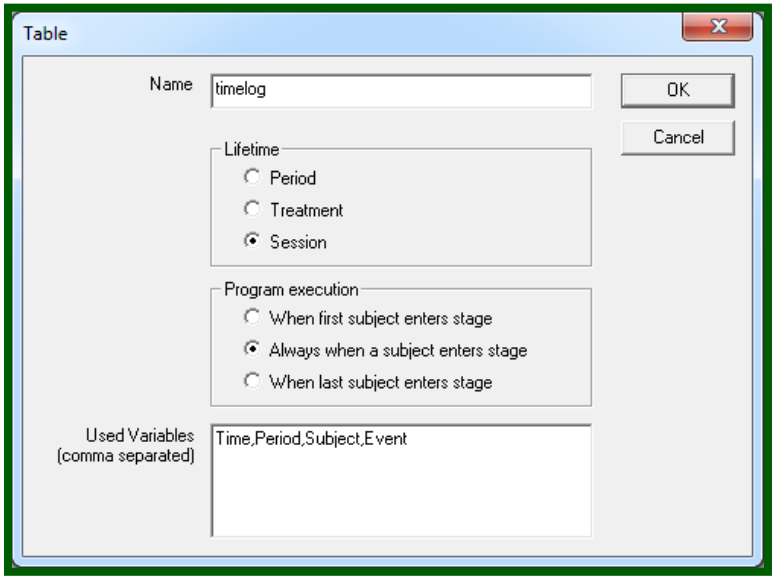
- Search and replace using export function
  - Use File-Export-Treatment to export code to a .txt-file.
  - Use search/replace in Notepad/Word…
  - Re-import treatment file into z-Tree
- Calculate exact time using gettime ()
  - gettime() returns time since computer was started
  - Set variable equal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

Timelog

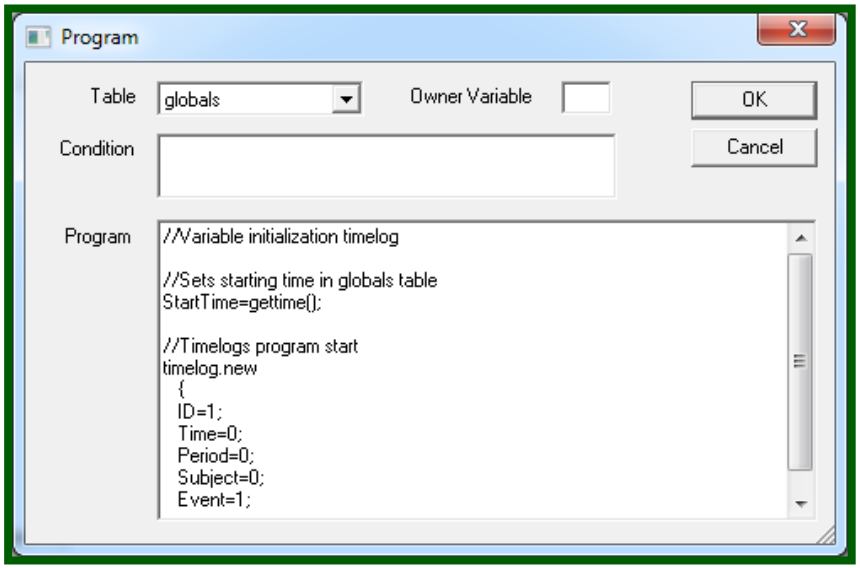
- Record exact timing of all events in the program:
- Calculate exact time using gettime ()
  - gettime() returns time since computer was started
  - Set variable equal to gettime() at beginning of period
  - Calculate difference between time of interest and time calculated at the beginning

Timelog

- Create timelog table:
  - Allows custom logging of important events

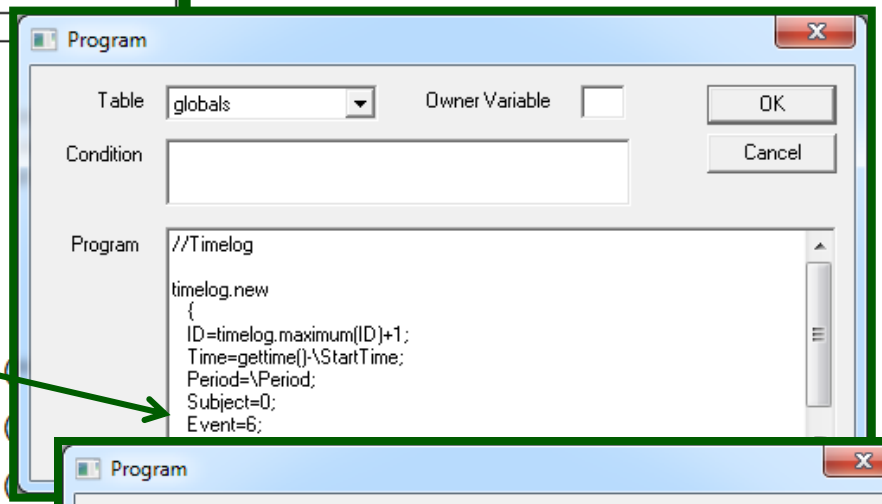


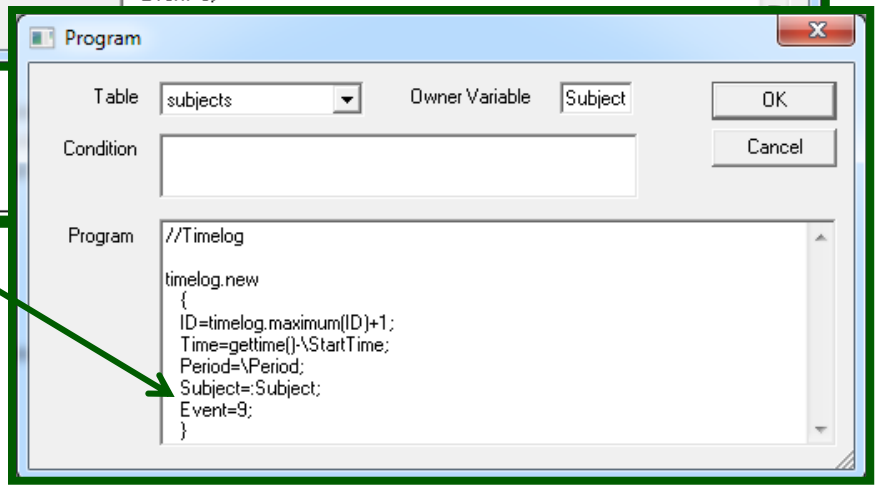
- Record starting time and create first record:



- 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 (12 Continue from screen ControlQuestion2 (4) 13 Return to previous screen from ControlQuestion2 (14 Continue from screen ControlQuestion3 (5) 15 Return to previous screen from ControlQuestion3 (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

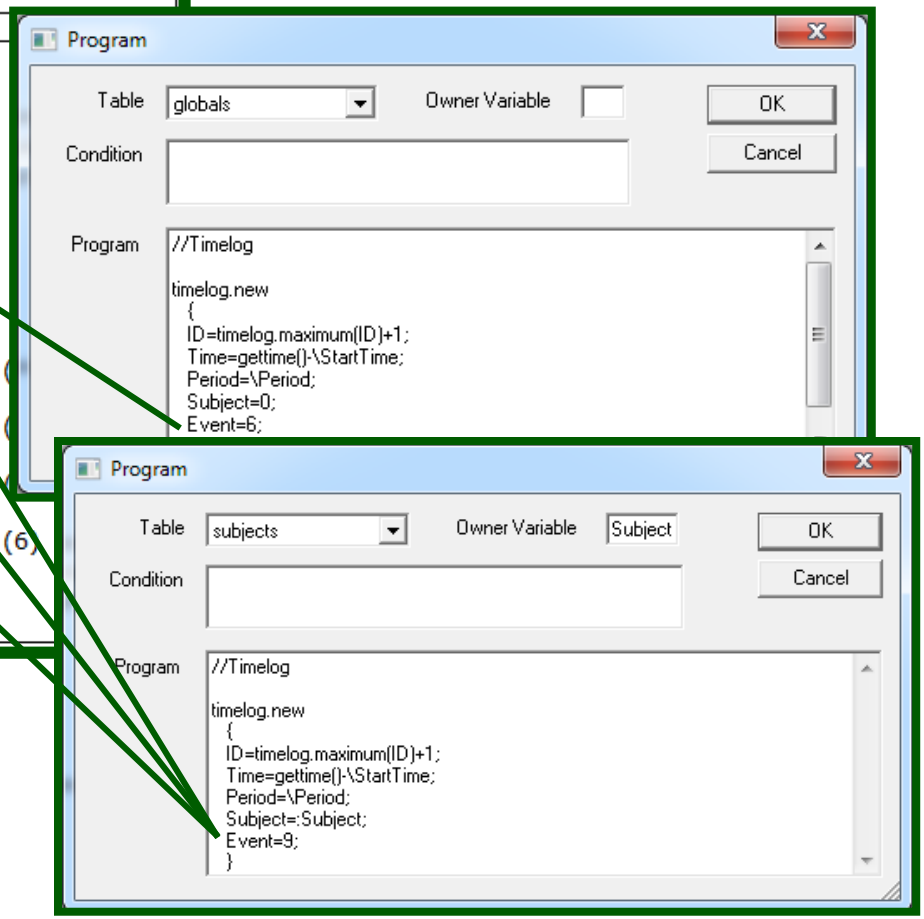




- Create entries for all events (of interest):

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



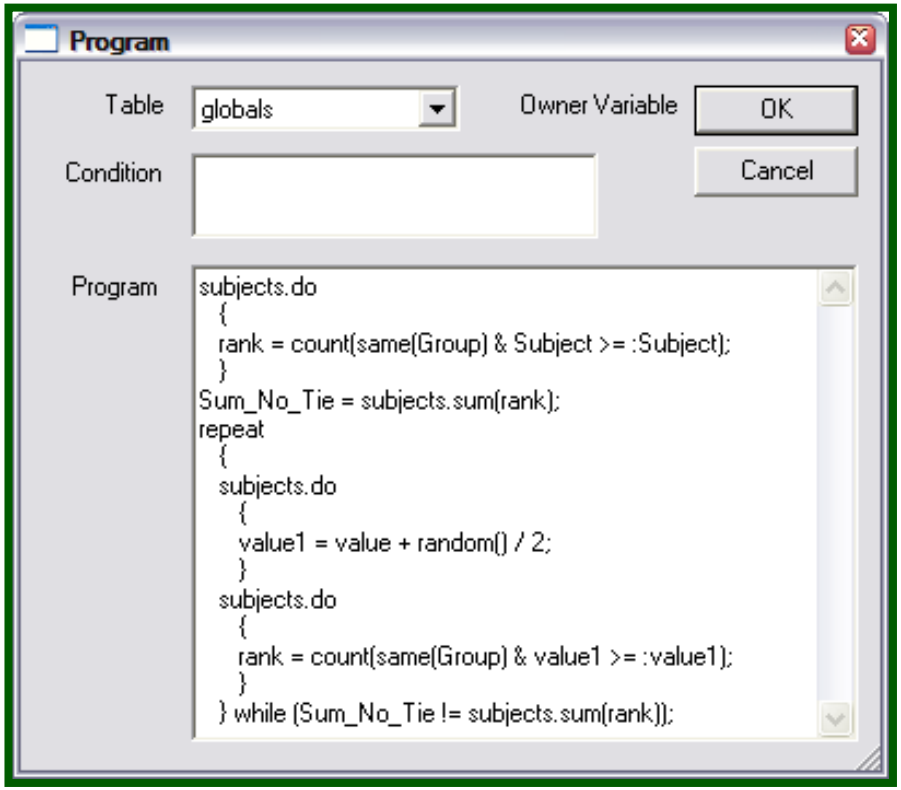
Experimenter subject*

- 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

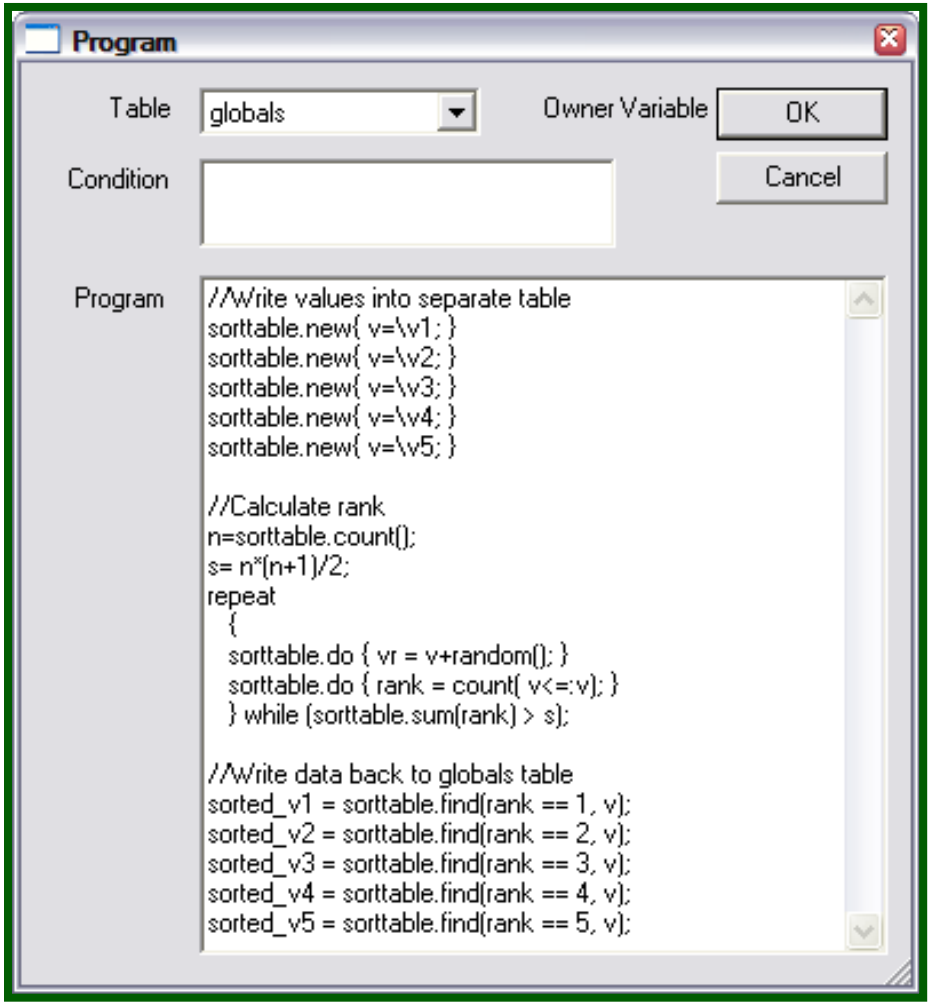
- Simple rank calculation (grouping is optional):
  - rank_low = count (same (Group) & value > :value) + 1;
  - rank_high = count (same (Group) & value >= :value);
- Rank calculation excluding ties (assuming value is integer):



Source: z-Tree Wiki.

Sorting values in z-Tree*

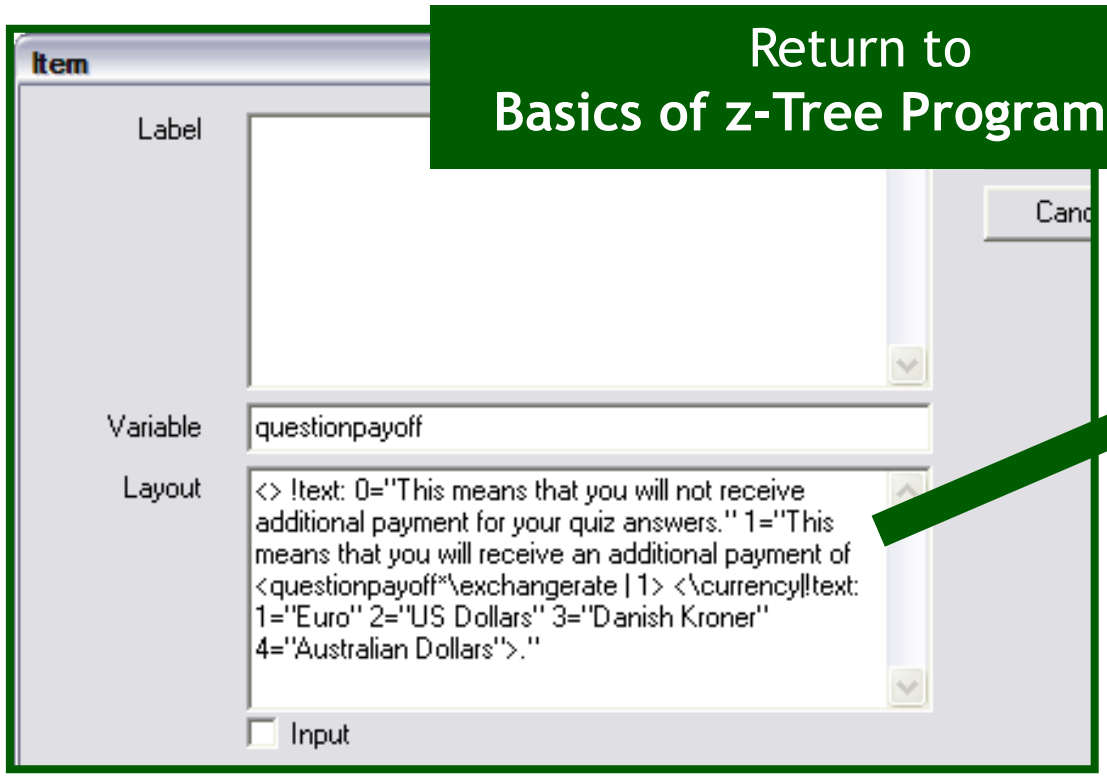
- Assume there are integers v1 to v5 in the globals table
- Strategy:
  - Put data into a table
  - Calculate the rank
  - Transfer the data back



* 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…

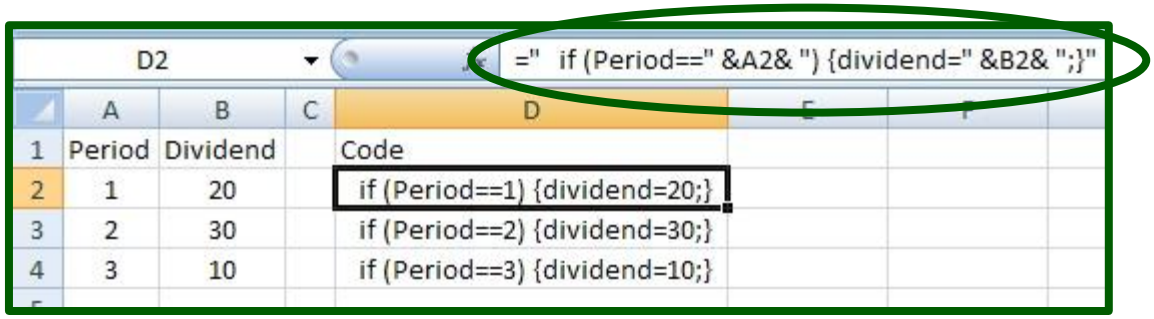


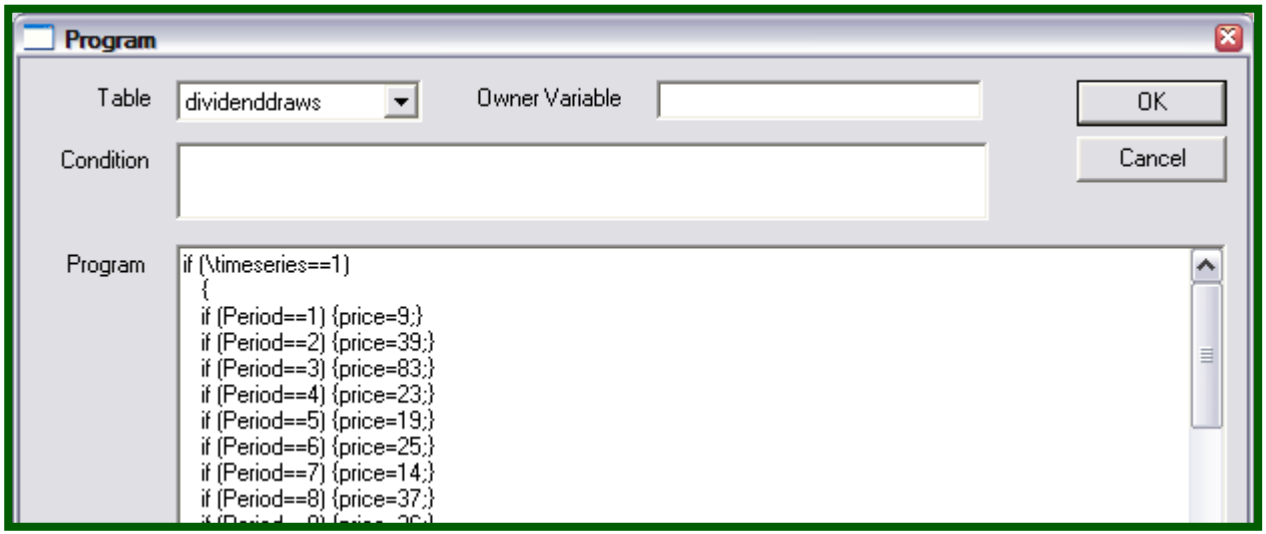
Return to
Basics of z-Tree Programming

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:

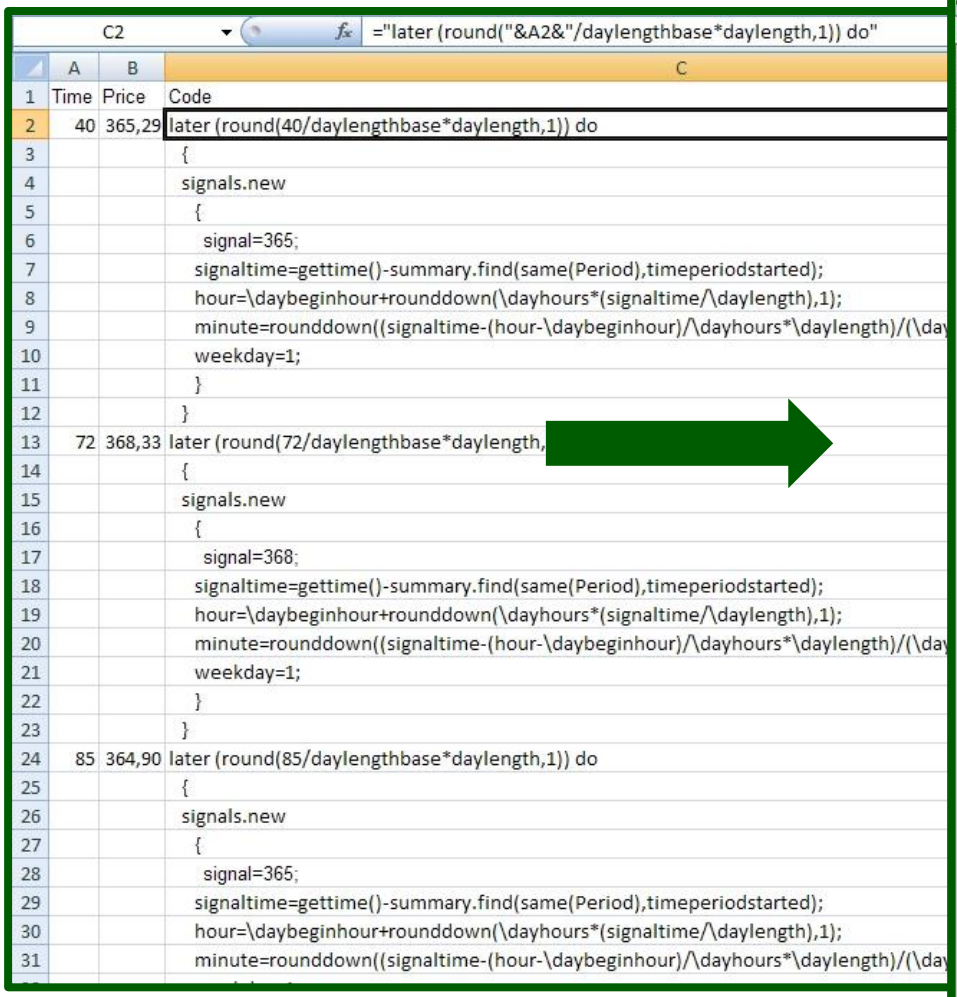




Generating repetitive code with Excel

- Complex code generation

Return to
Import and Export



C2: ="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

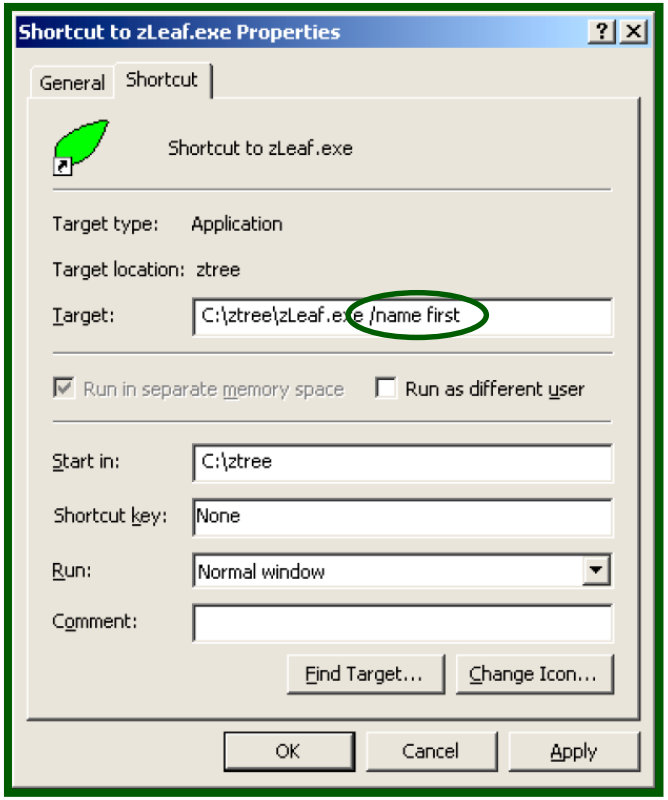
zTree - [#daynight_baseline_overnight.ztt]

File Edit Treatment Run Tools View ?

```
StartingScreen =|= (-1)N
TradeScreenWE =|= (welength)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:

killleafs.bat - Editor

Datei Bearbeiten Format Ansicht ?

```
TASKKILL /F /IM "zleaf.exe"
```

- killtrees.bat shuts down all running z-trees (process often remains active after closing z-tree):

killtrees.bat - Editor

Datei Bearbeiten Format Ansicht ?

```
TASKKILL /F /IM "ztree.exe"
```

- killplants.bat shuts down both:

killplants.bat - Editor

Datei Bearbeiten Format Ansicht ?

```
TASKKILL /F /IM "ztree.exe"
TASKKILL /F /IM "zleaf.exe"
```

Return to
Public Goods Game Tutorial

- Note that growleafs.bat automatically shuts down any remaining z-leafs before starting them anew:

growleafs.cmd - Editor

Datei Bearbeiten Format Ansicht ?

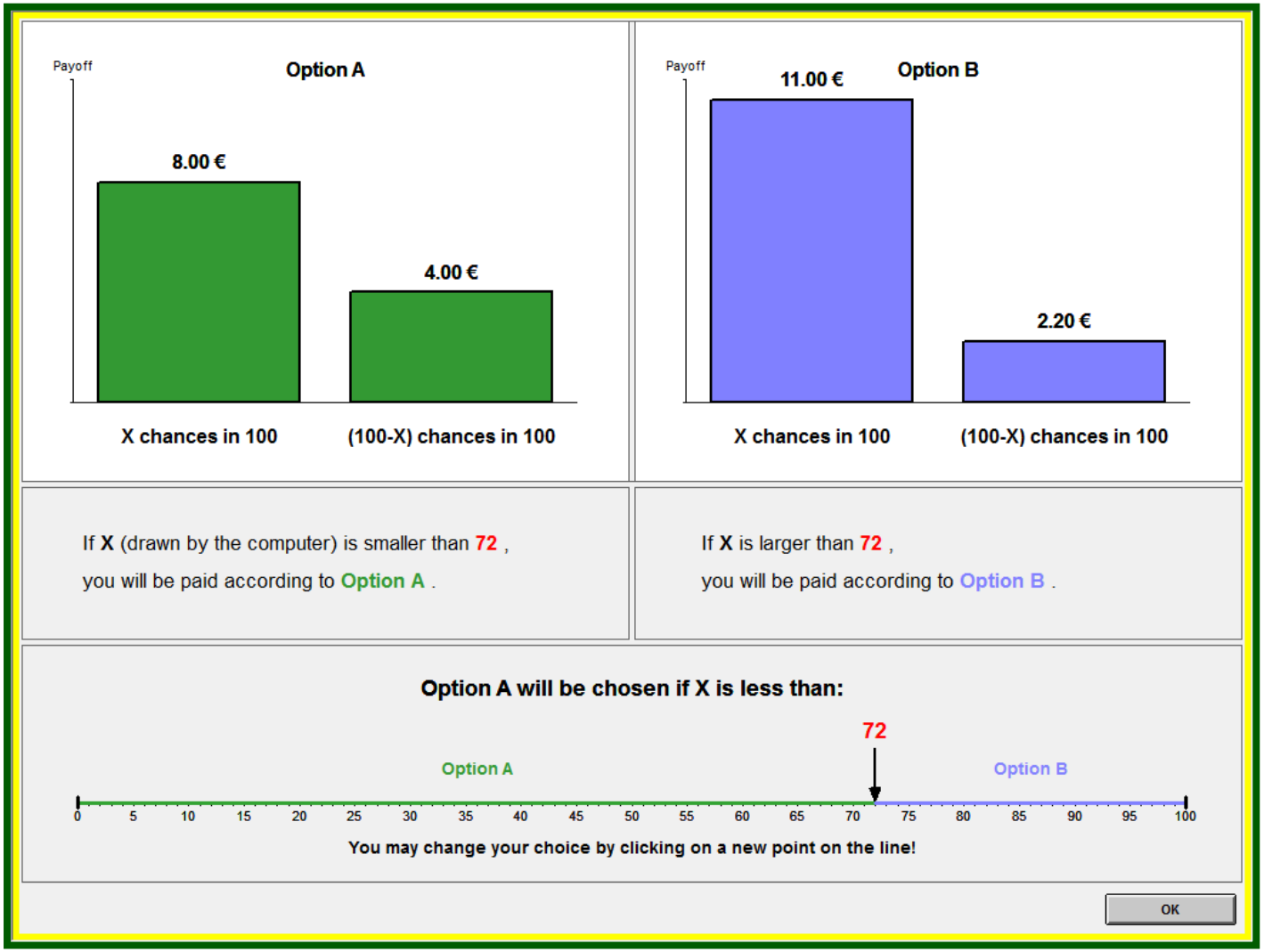
```
@echo off
SETLOCAL ENABLEEXTENSIONS
SET /P leafs=How many zleafs do you want? %=%
cls
TASKKILL /F /IM "zleaf.exe"
del server.eee
for /L %%X IN (1,1,%leafs%) do (
start zleaf.exe /size 1024x768 /name Subject%%X% /fontsize 11 /position 20,20
PING 1.1.1.1 -n 1 -w 10 >NUL)
```

Graphics Teaser

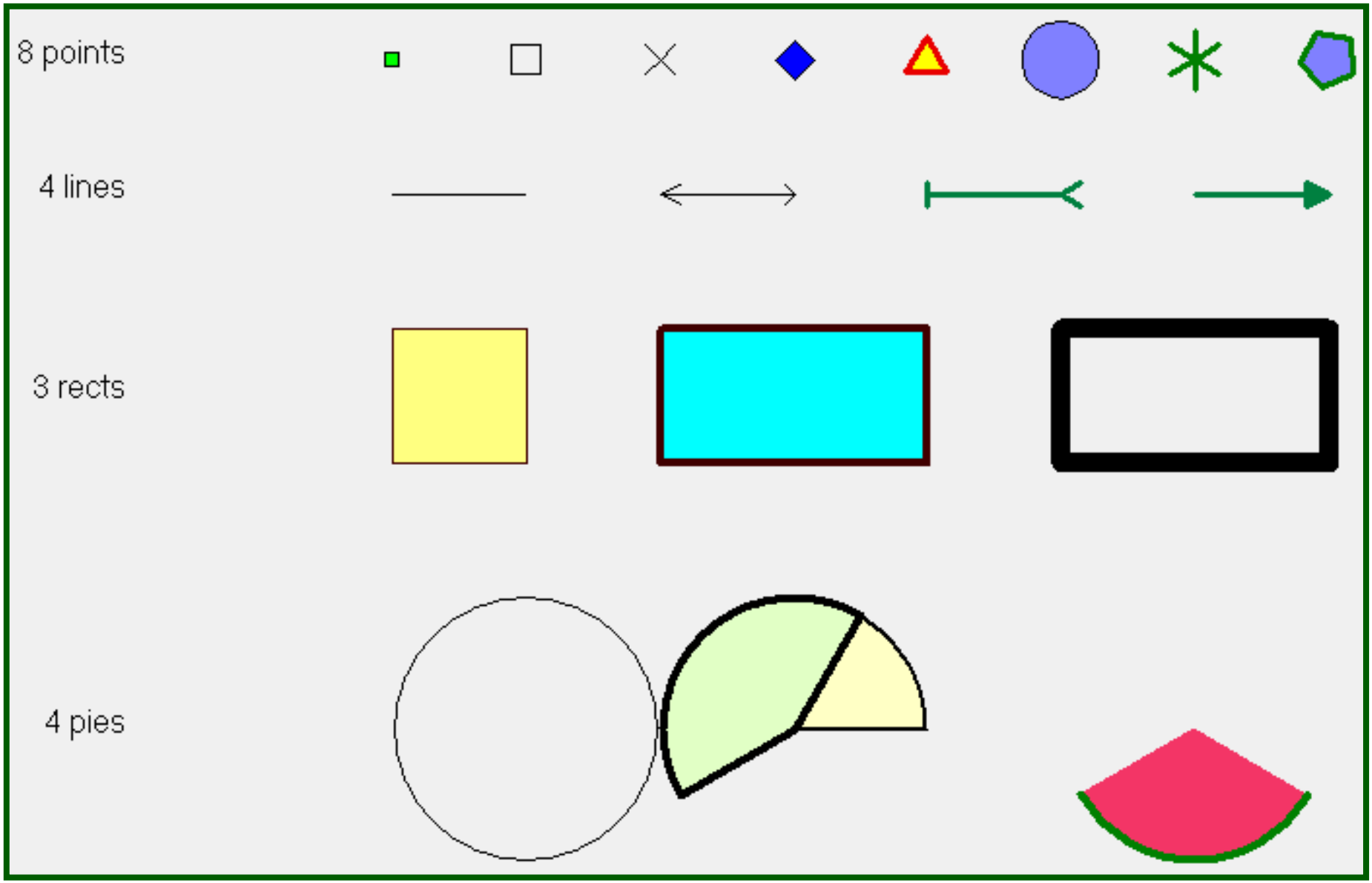
- Capabilities
- Plot items
- Demo treatments

Graphics teaser

- z-Tree offers graphics capabilities since version 3.0



Selection of plot items



Further information

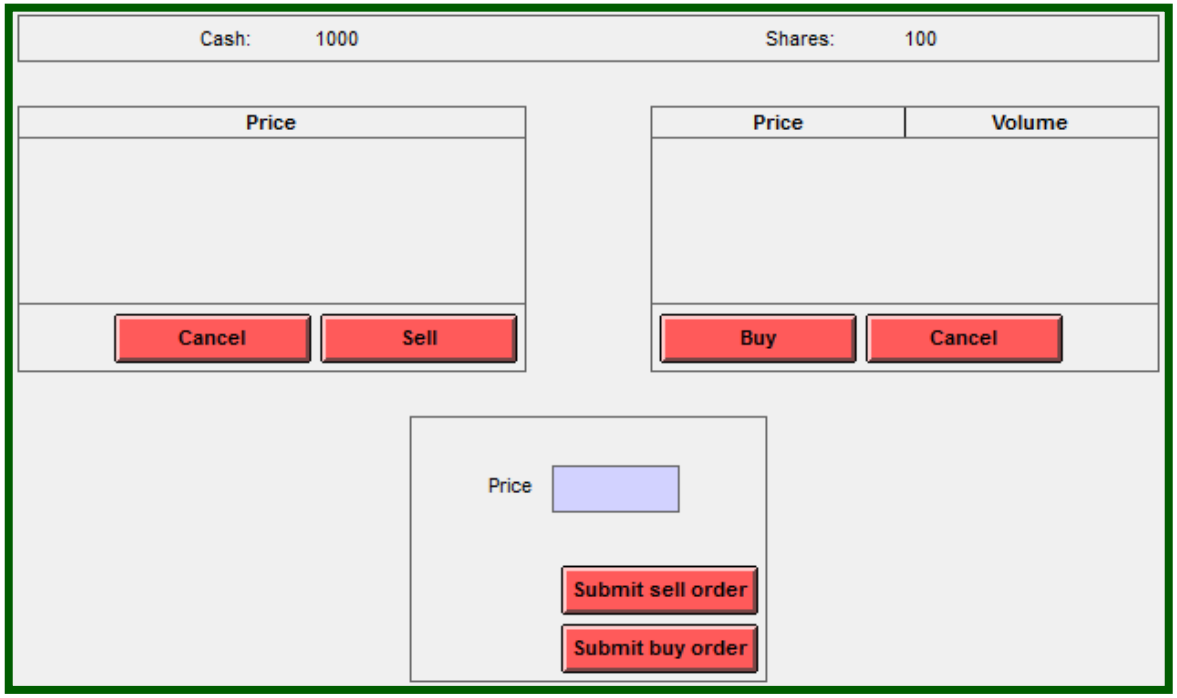
- Demo treatments demonstrate graphics capabilities:
  - colortriangle.ztt
  - movepointdemo.ztt
  - selectlinkdemo.ztt
  - z-draw.ztt
  - animatetreedemo.ztt
- More information on graphics in z-Tree:
  - z-Tree lecture by Maria Bigoni: http://www2.dse.unibo.it/bigoni/courses/ztree/Lecture_notes_handout.pdf
  - 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



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.

Markets and auctions: contracts table*

- Buttons in contract creation box and contract list box
  - Can contain checkers (e.g. price * volume < 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:
  type == 1 & status == 0
- Open sell offer characterized by:
  type == -1 & status == 0
- Transaction characterized by:
  status == 1

Concept: Contracts table record

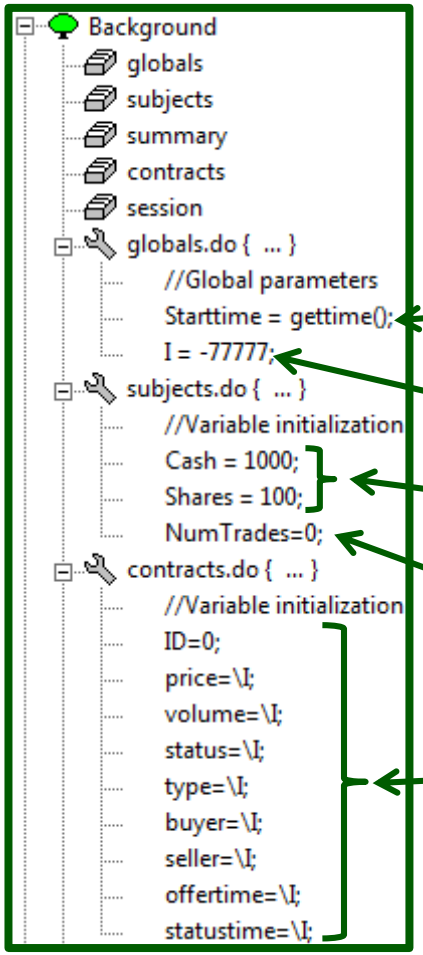
- Example records:

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 intialization:



Record starting time for calculation of time elapsed

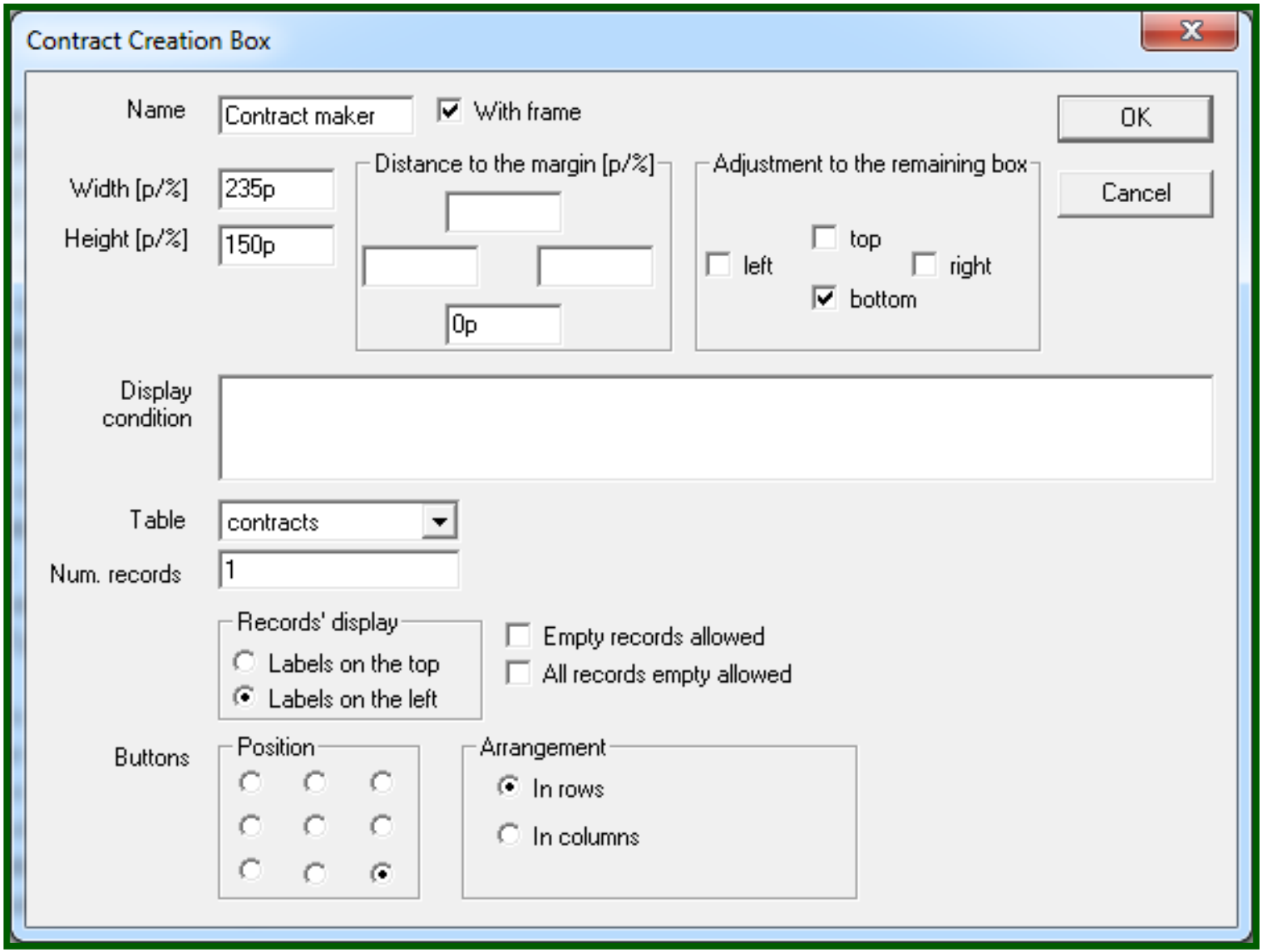
Set dummy number for variable initialization

Set endowment

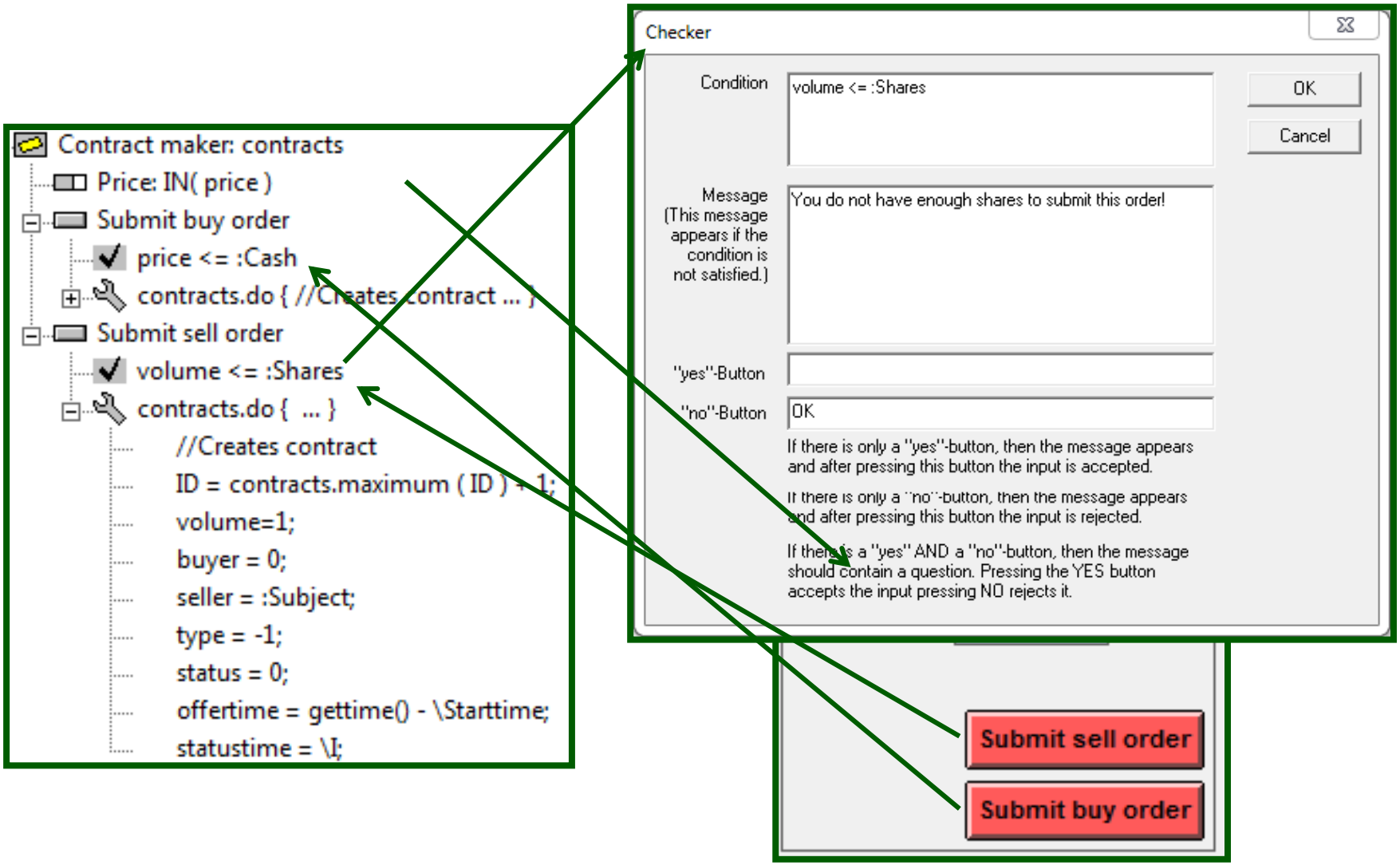
Initialize variable for convenient calculation of trade number

Initialize variables

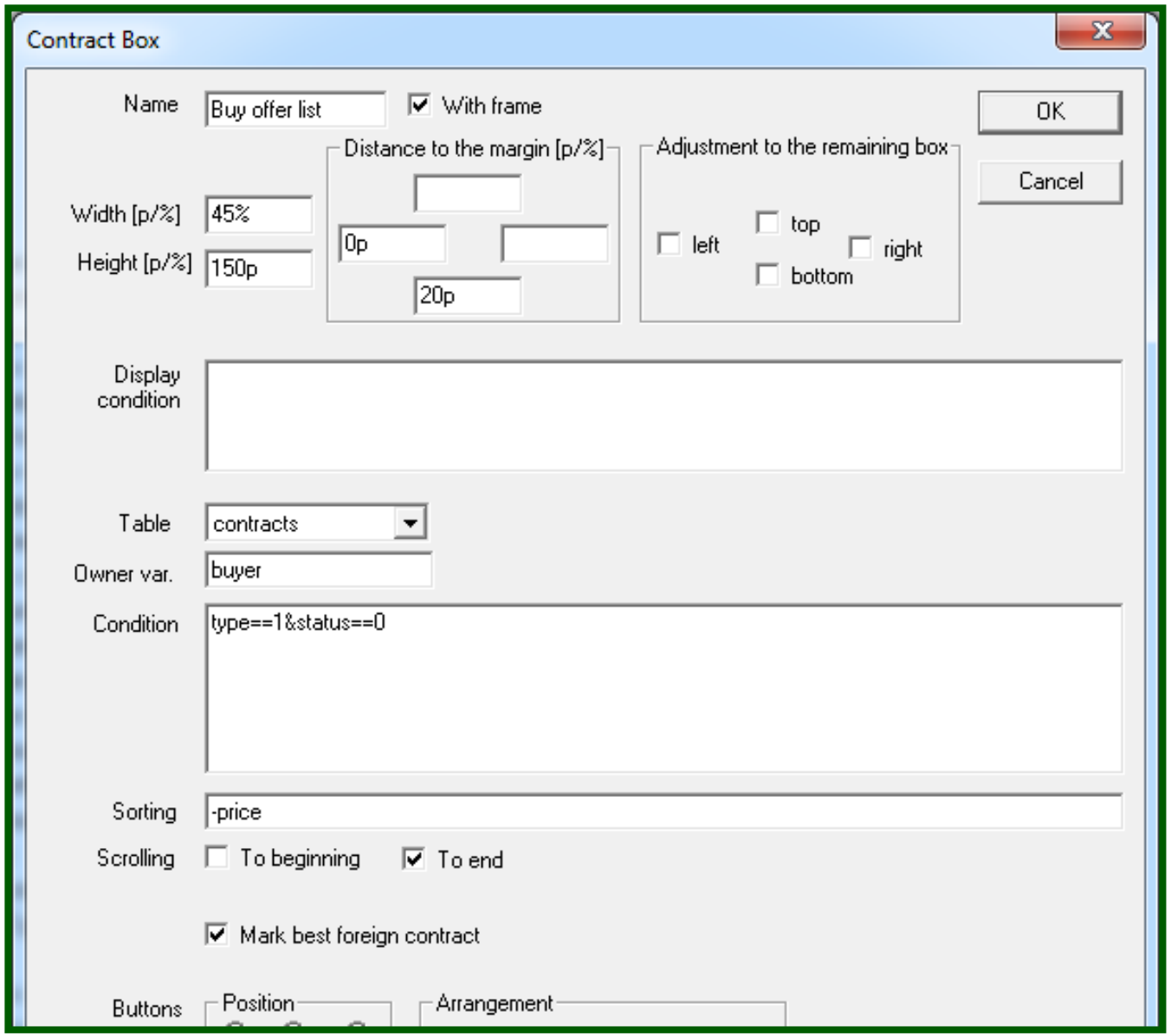
Limit order input



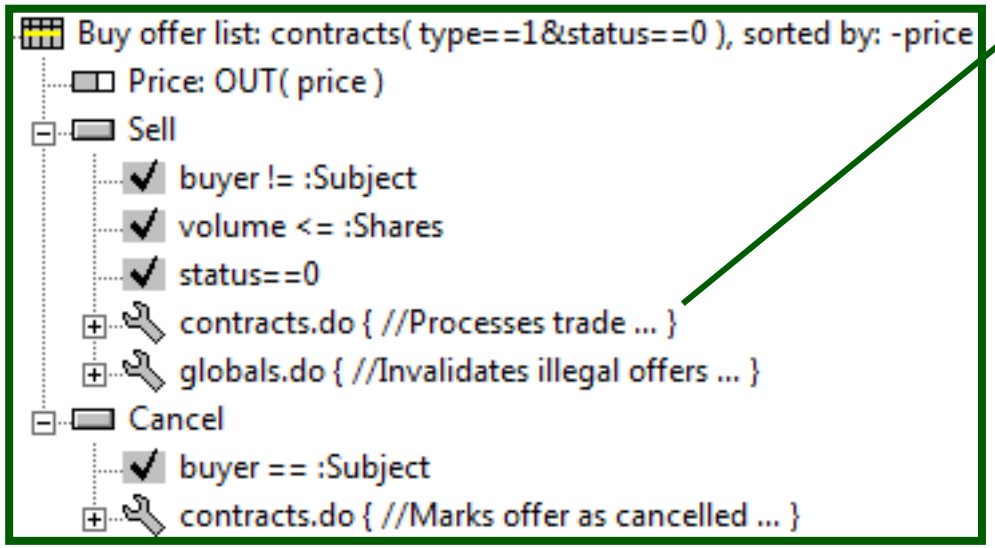
Limit order input



Order display

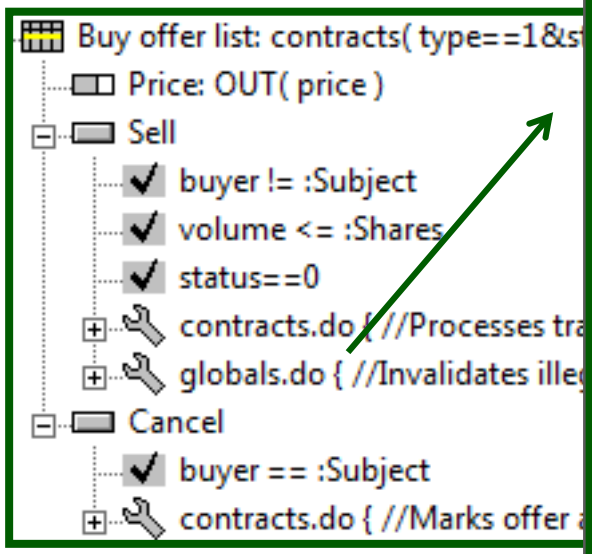


Market order submission



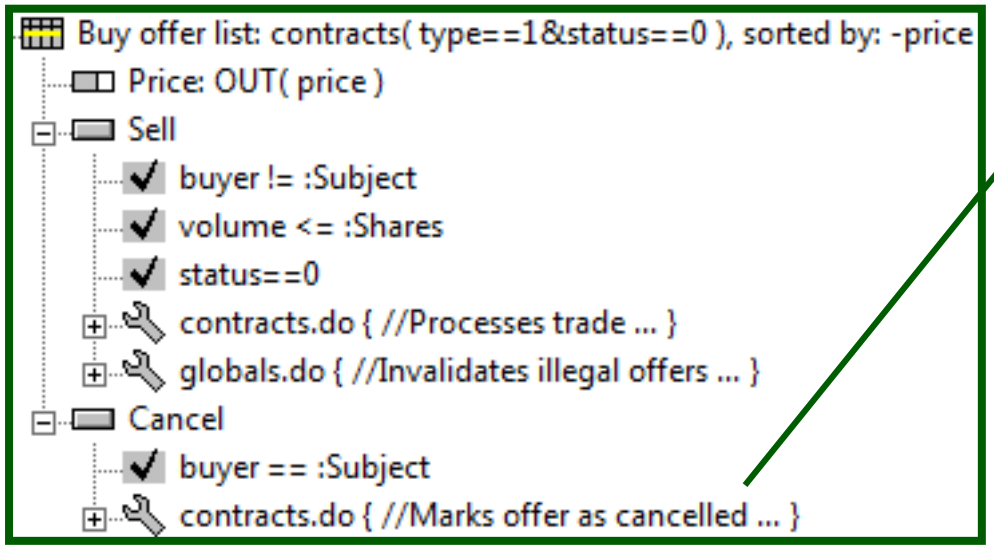

```
//Processes trade

//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



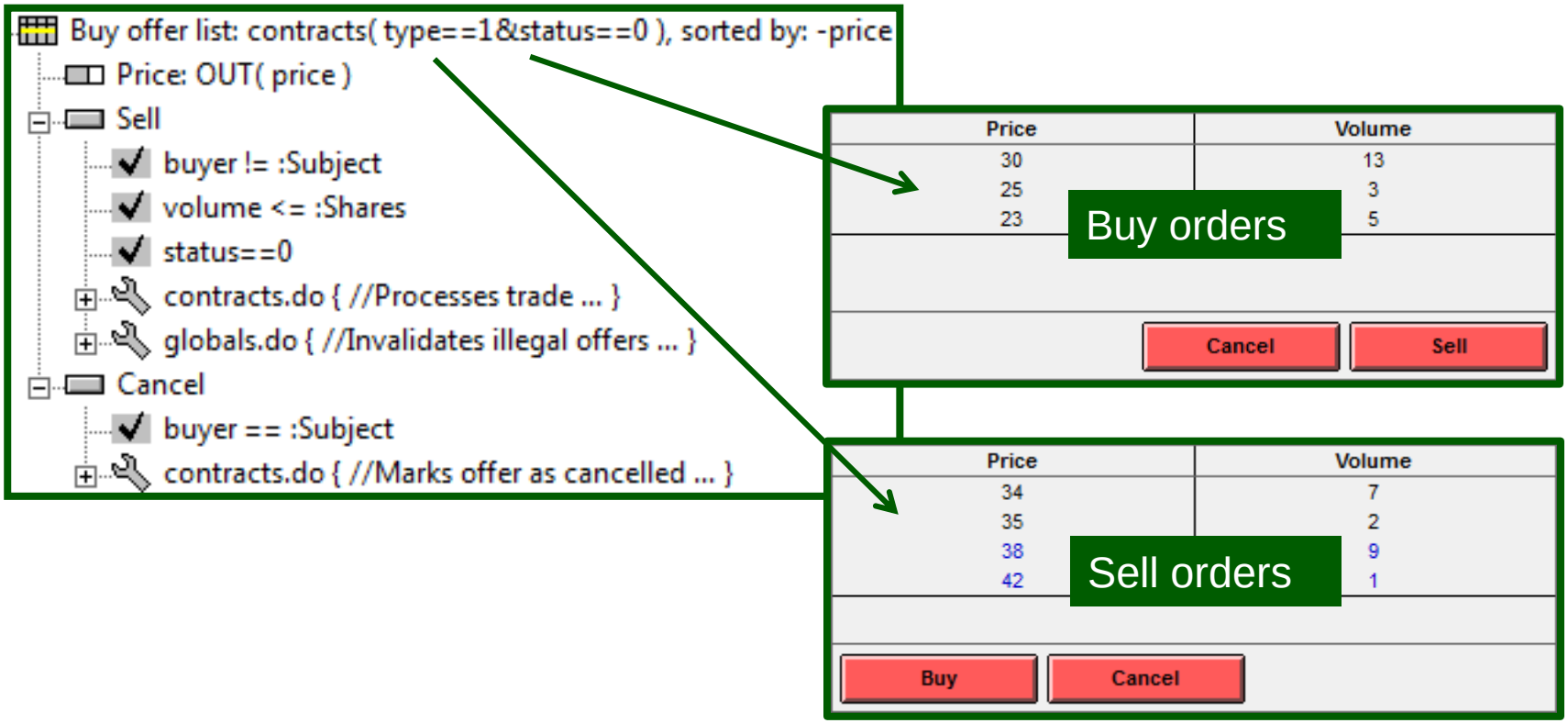

```
//Invalidates illegal offers

//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



Limiting order legality

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

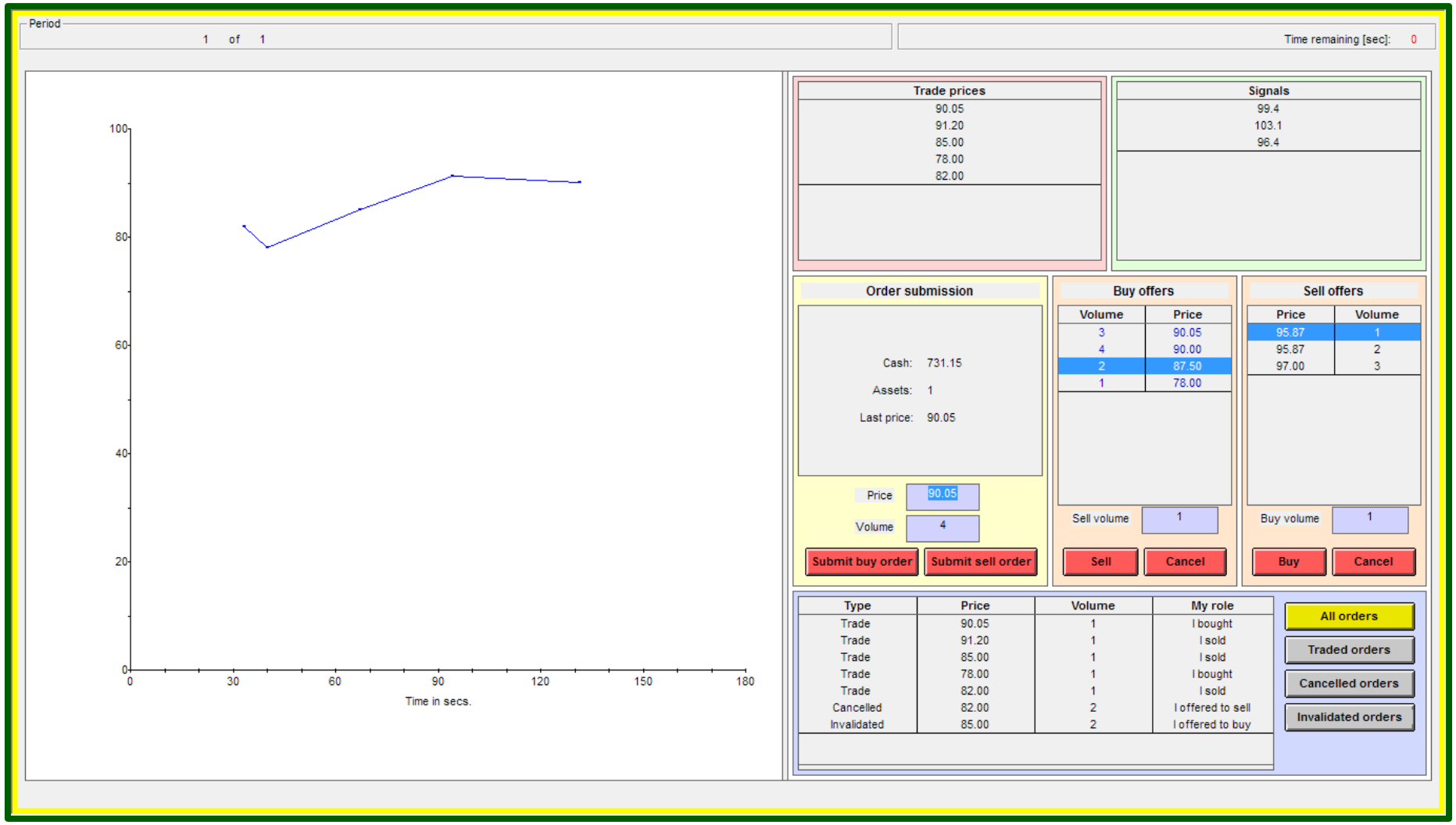
Concept

- 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



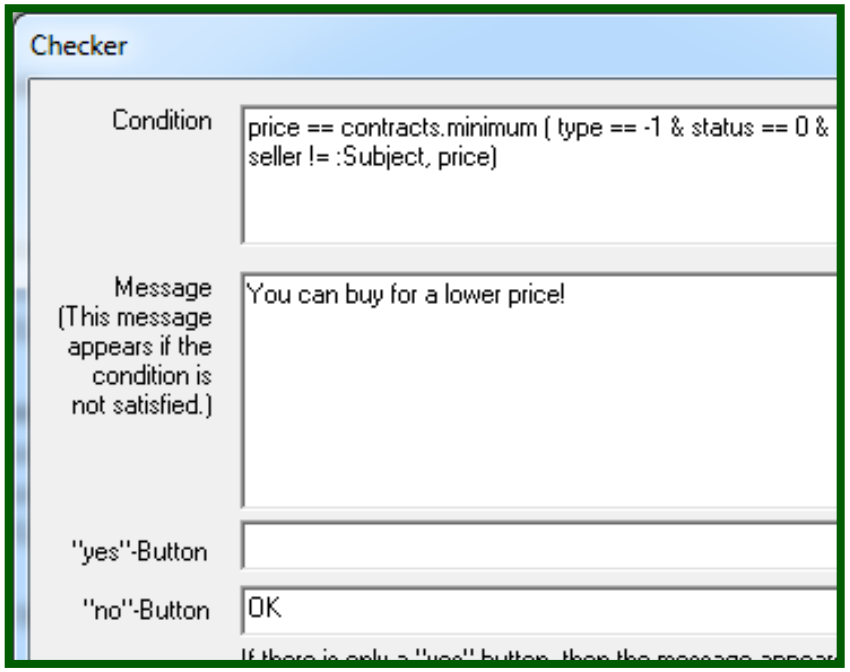
Source: GIMS.

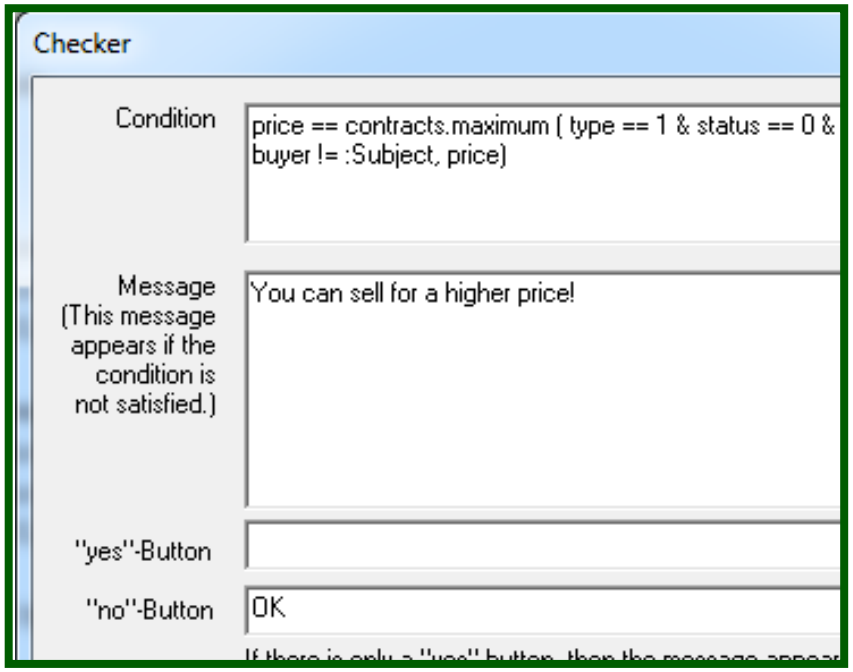
Order Legality

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

Transacting best outstanding order

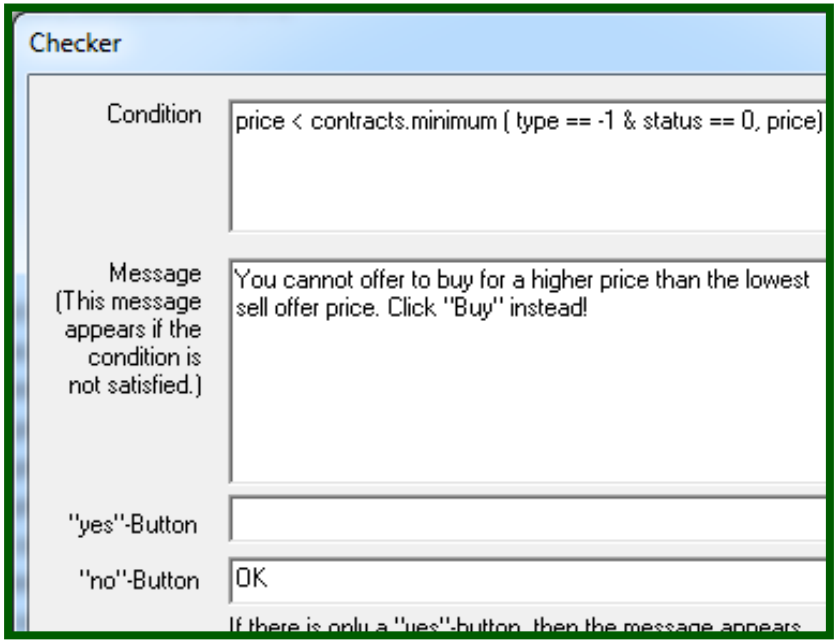
- Write down the code for a rule to ensure that the subject buys (sells) for the lowest (highest) price when accepting an outstanding offer.

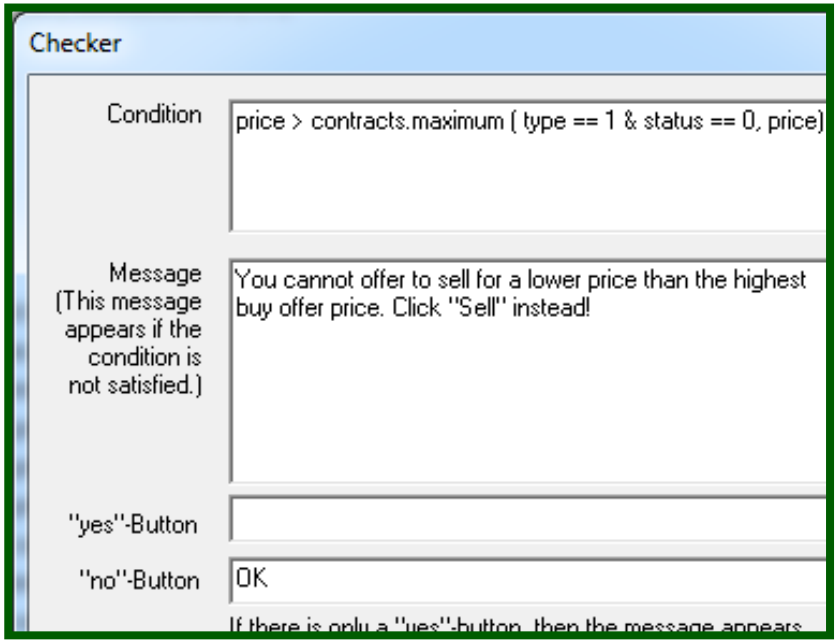




No-crossing rule

- 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.

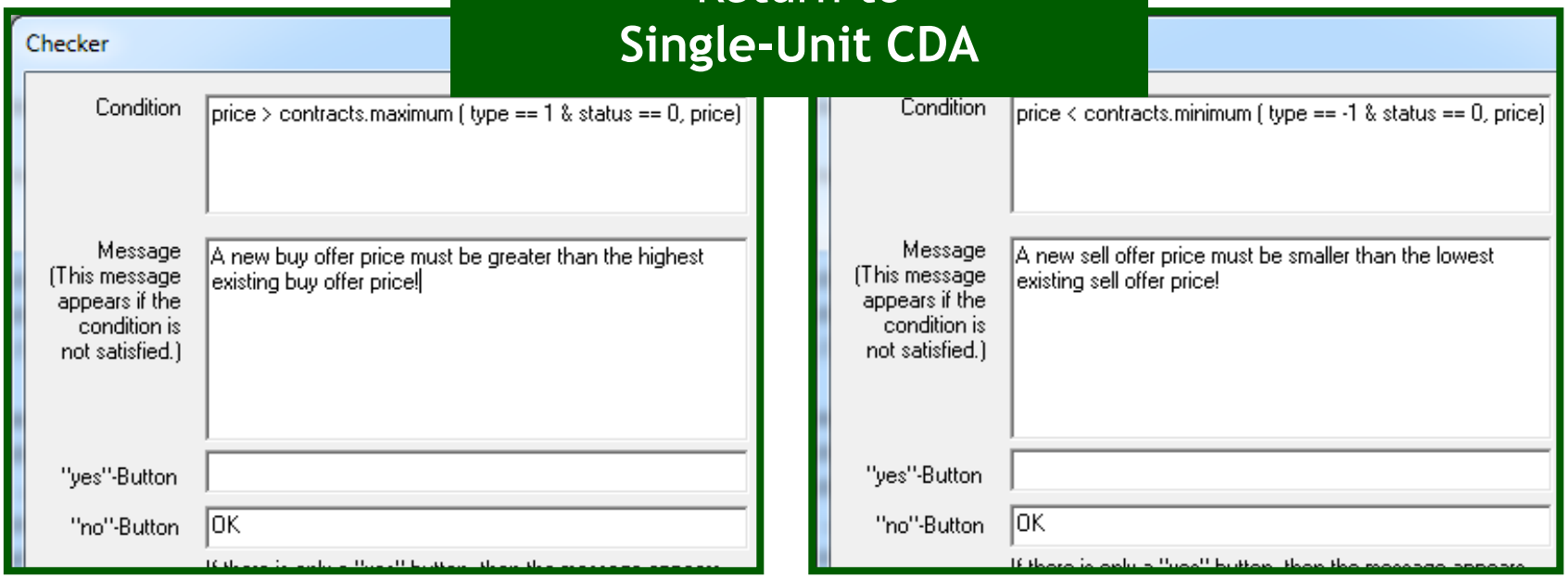




Bid-ask improvement rule*

- 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.)

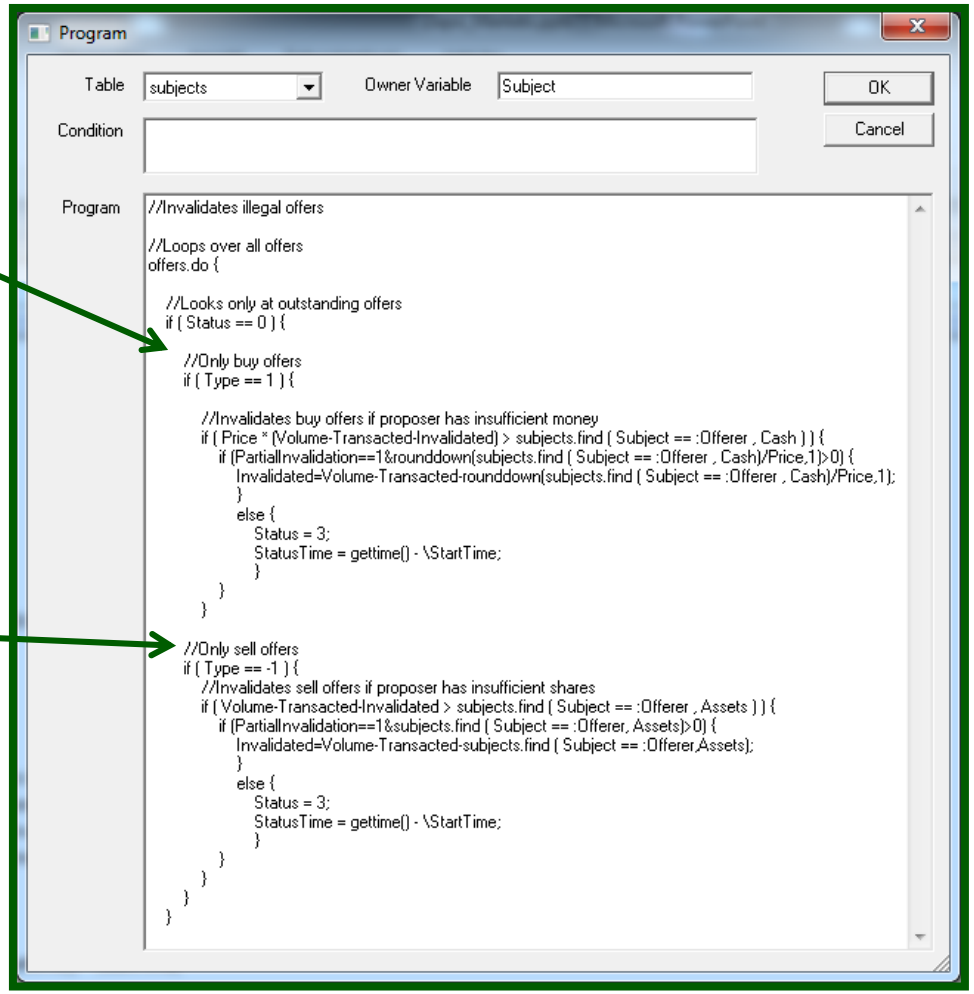
Return to
Single-Unit CDA



* 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



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

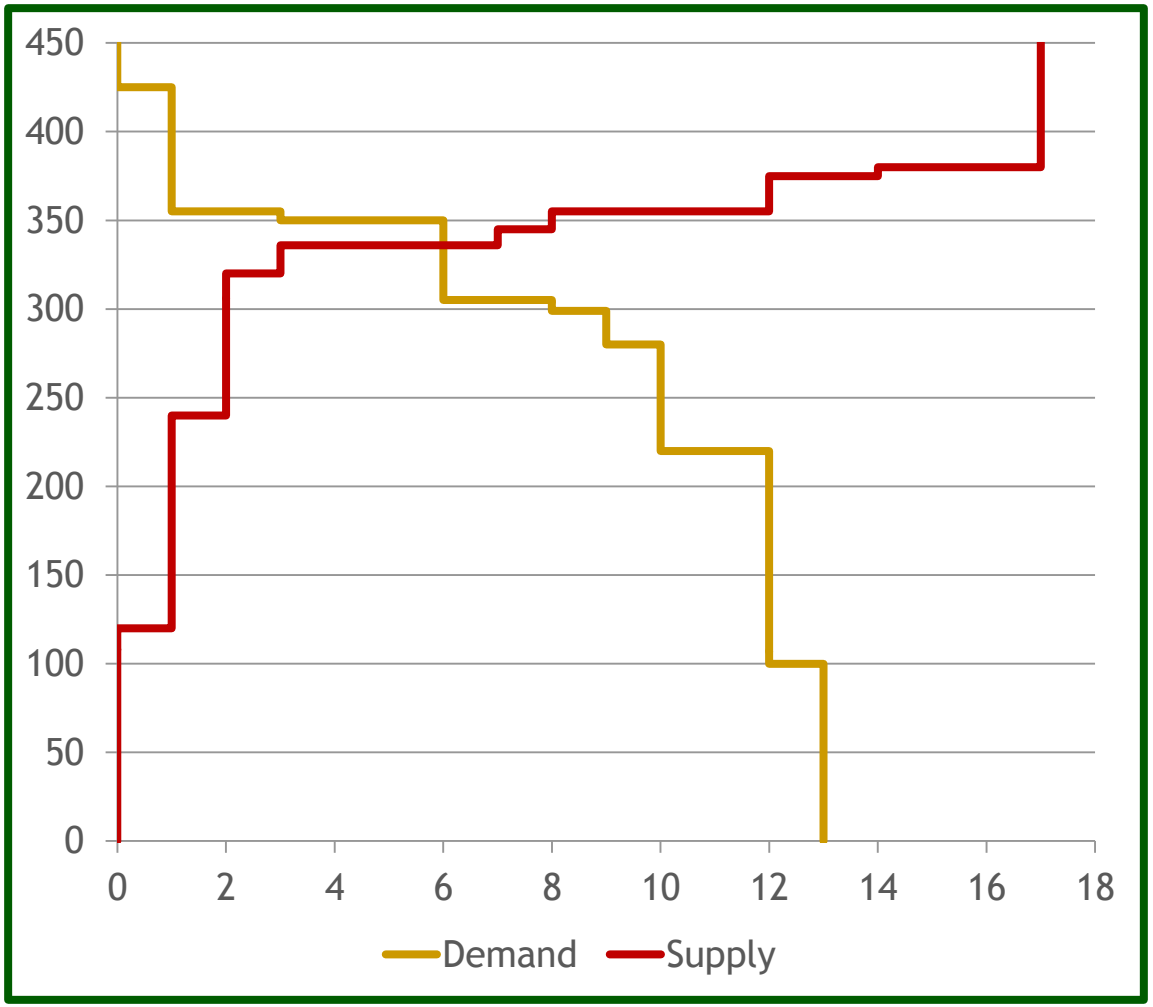
Effective programming

- 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(Volume)
  - Mark orders as (partially) transacted
  - Update buyer's/seller's cash and asset balances
  - Repeat

Order matching procedure

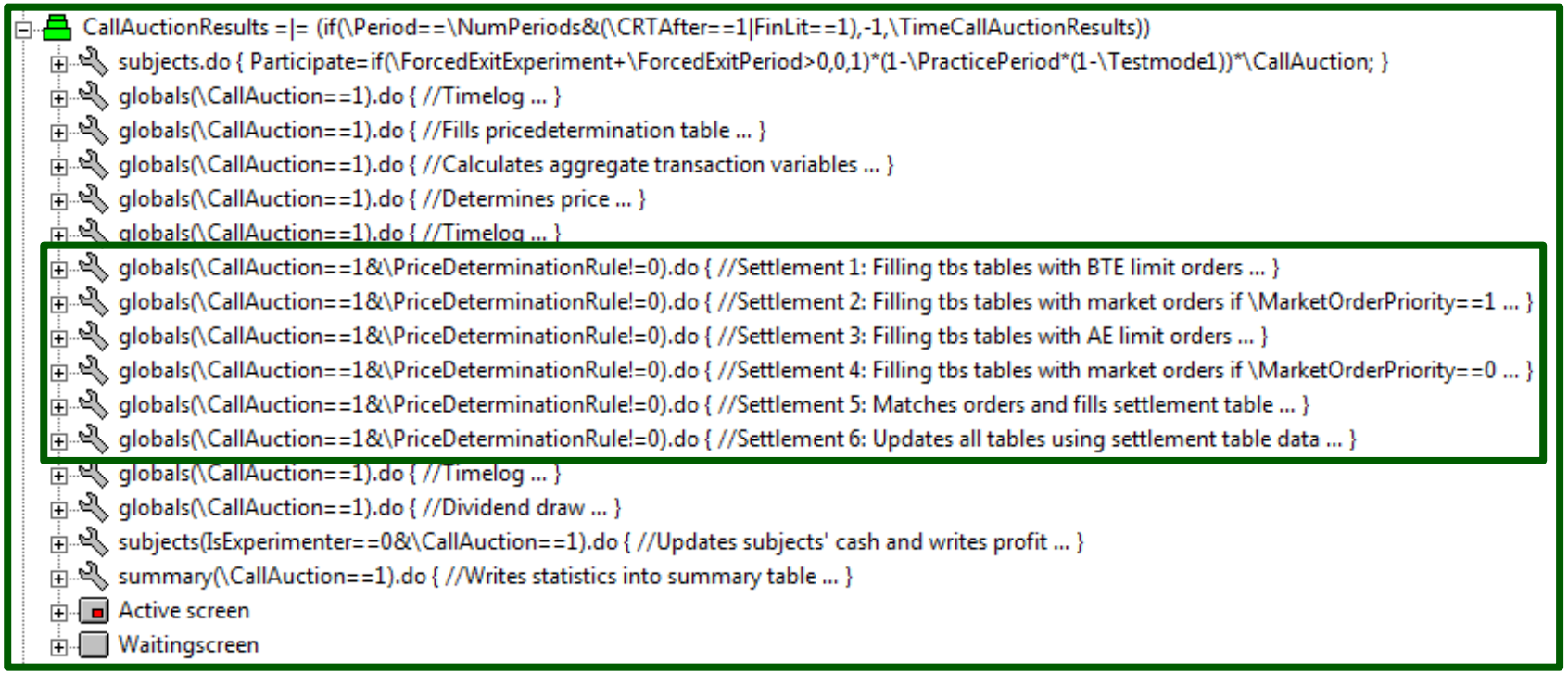
- Solution 1: Use no extra tables (only offers and contracts)
  - See GraphSettlementExtensive.graphml
  - Everything needs to be done in a single program
  - Always needs to loop through all records in offers table
  - Very inefficient

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)



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=roundup(random()*10,1);
  - Profit=if(x==y,100,0);

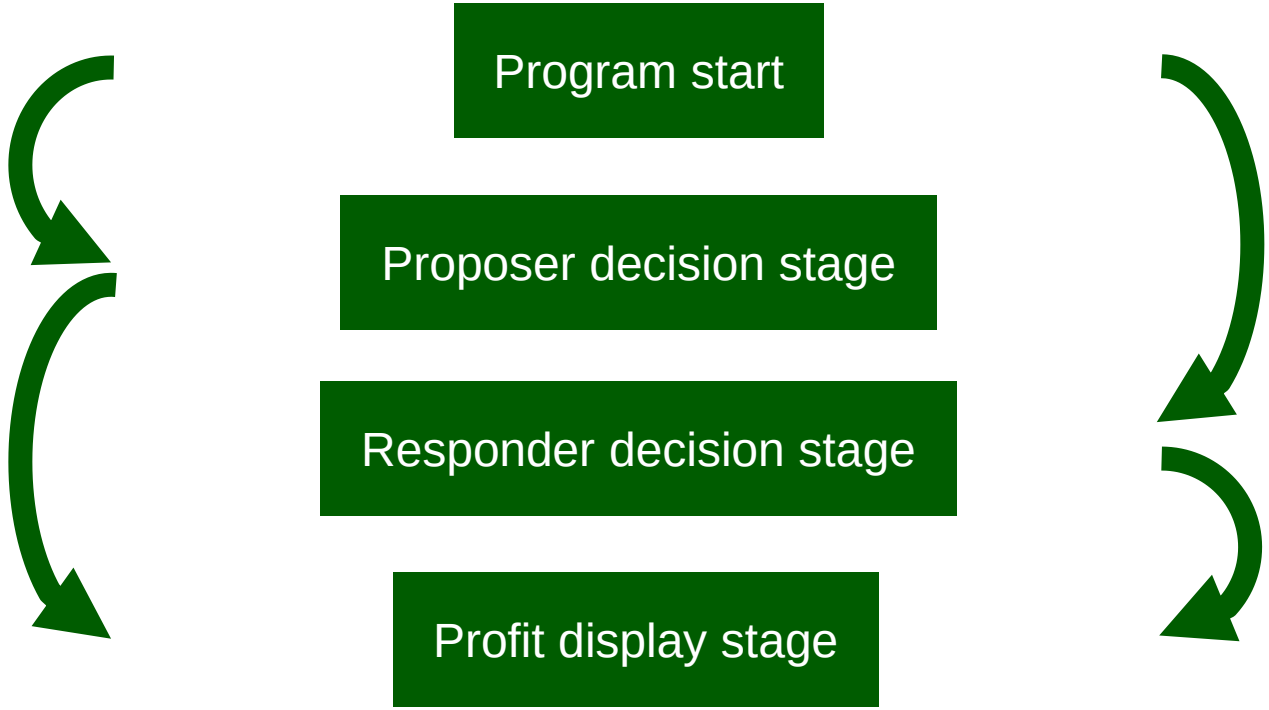
* Adapted from lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

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: Participate = 1 - 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

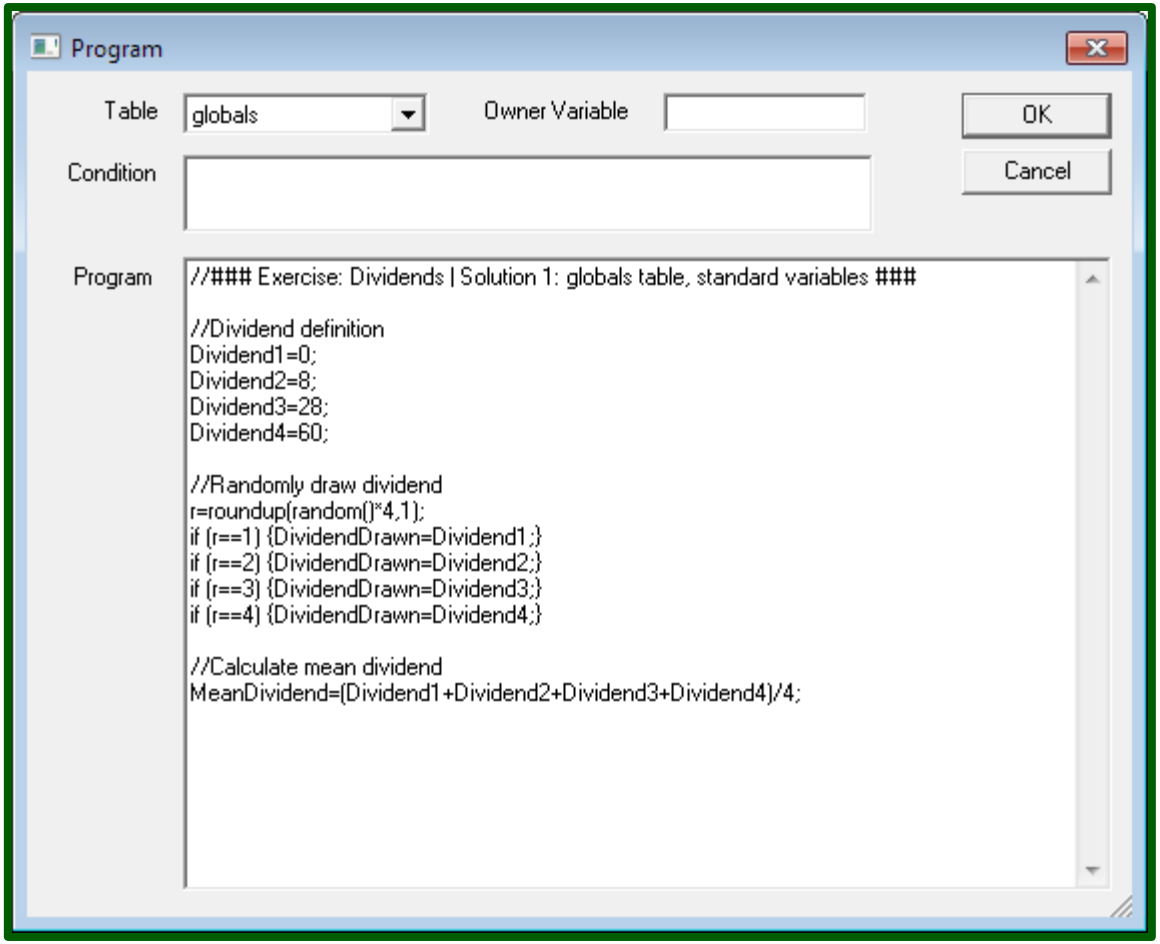
* Adapted from lecture slides by Verena Utikal, Friedrich-Alexander-University Erlangen-Nürnberg.

Dividends 1/5

- 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

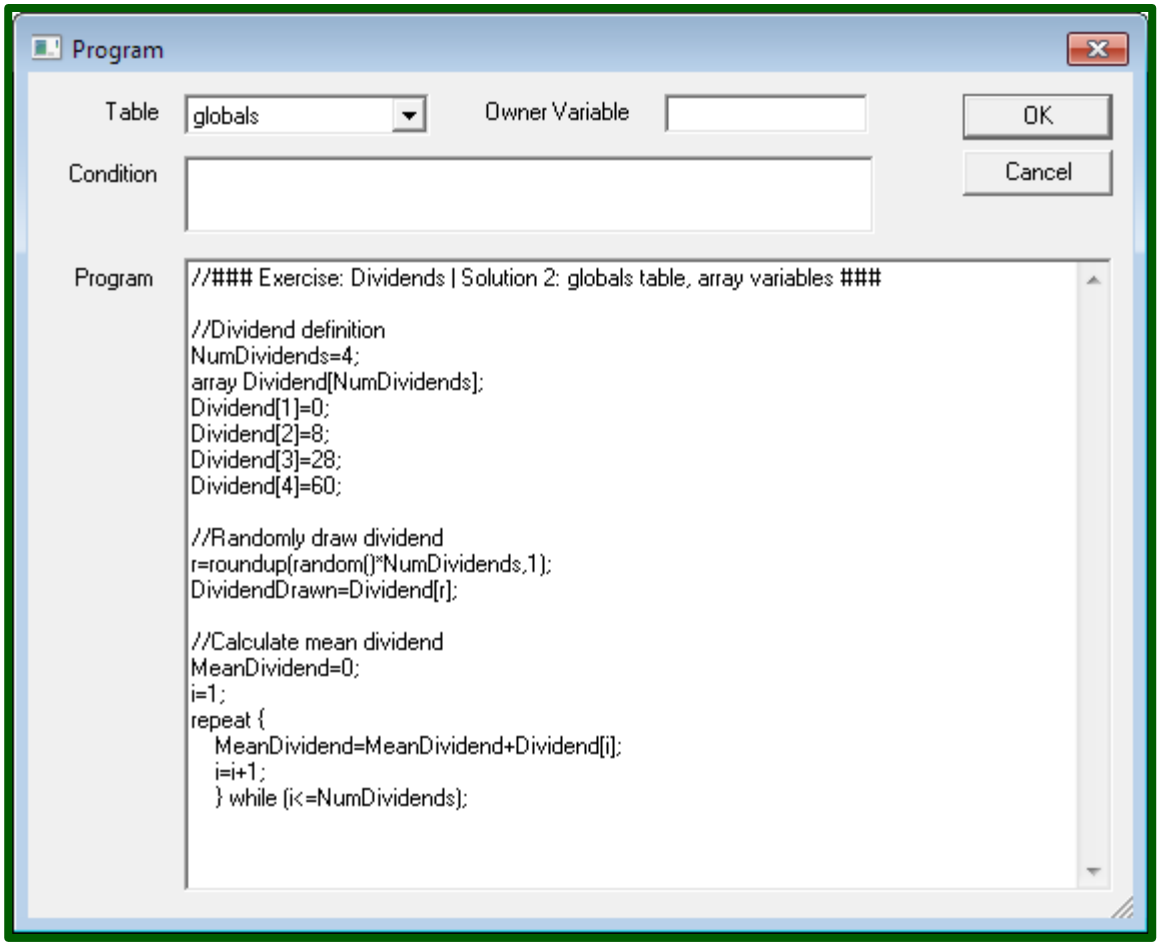
Dividends 2/5

- Solution 1: globals table, standard variables



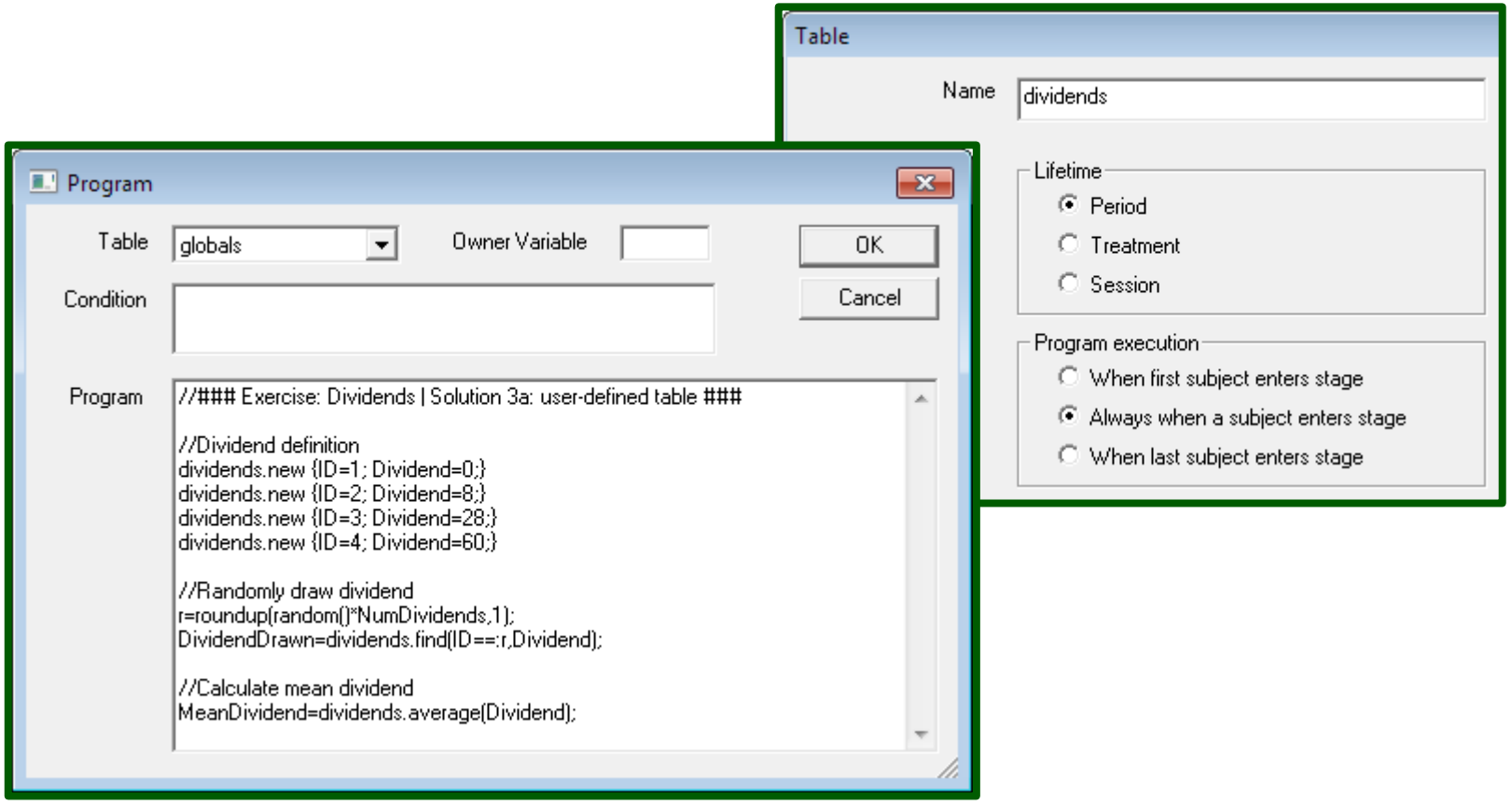
Dividends 3/5

- Solution 2: globals table, array variables



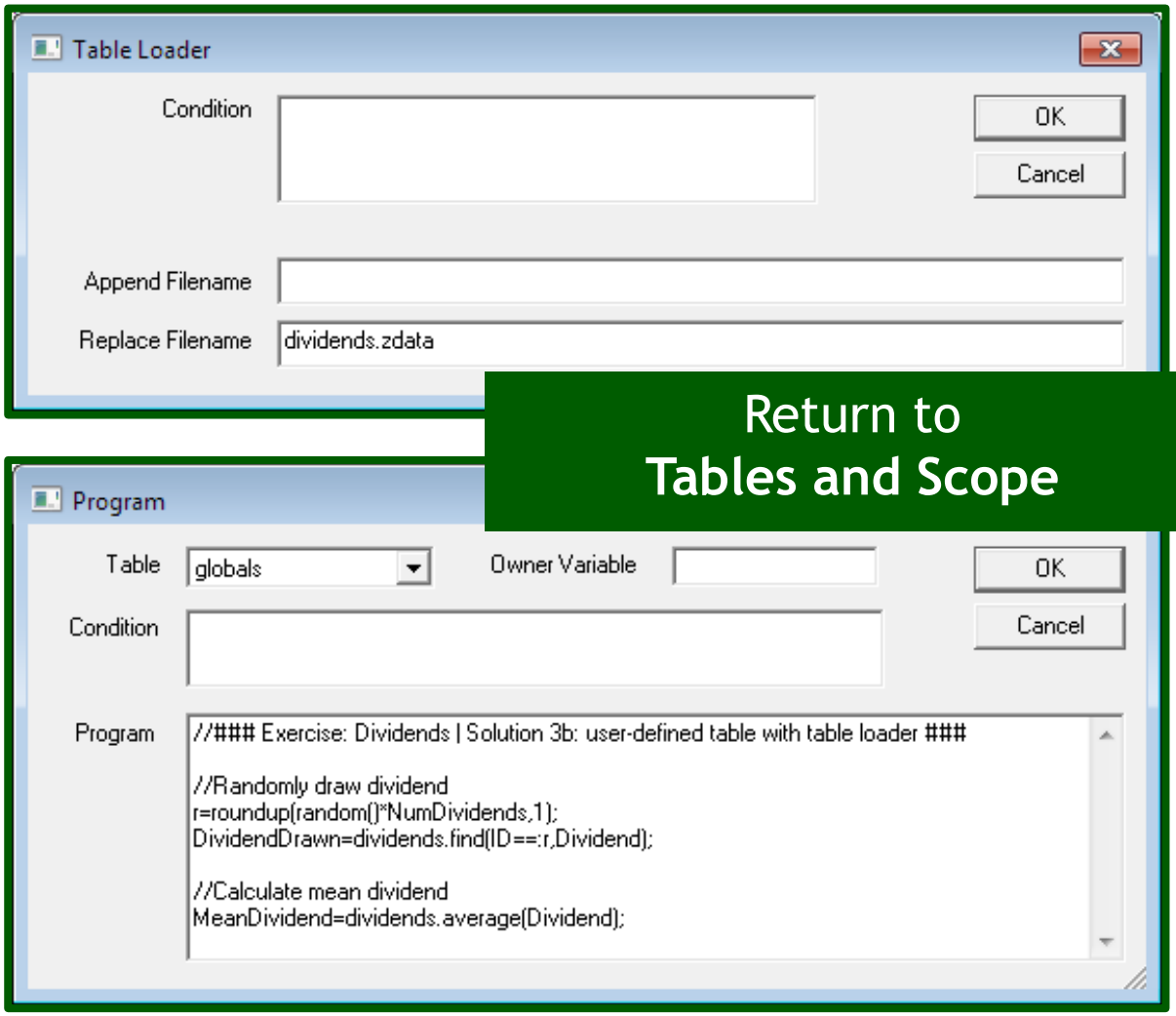
Dividends 4/5

- Solution 3a: user-defined table



Dividends 5/5

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



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

Return to **Subject Progression**

See CodeProgression.ztt

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