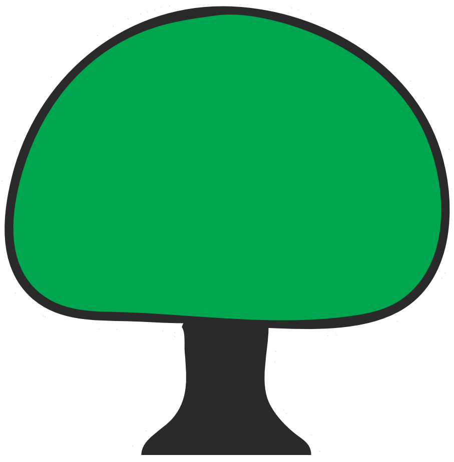
z-Tree



Zurich Toolbox for Ready-made Economic Experiments Urs Fischbacher

Ways to run a computerised experiment

researcher	→	assistant	→	computer
researcher	→	z-Tree		

- Using z-Tree removes the assistant as a source of misunderstandings.
- z-Tree can be used by computer illiterates.
- Also for computer literates:
  - z-Tree provides a comfortable and safe framework to implement experiments.

A Stage Tree

```
Background
    globals
    subjects
    summary
    contracts
    session
    subjects.do {
        EfficiencyFactor = 1.6;
        Endowment = 20;
        }
    Active Screen
        Header
    Waitingscreen
```

```
        ⊟··· Text
              :····  Please wait until the experiment continues
⊟··· Contribution Entry =|= (30)
:    ⊟··· Active Screen
:    :    ⊟··· Standard
:    :         :····  Your endowment:  OUT(Endowment)
:    :         :····  Your contribution to the project:  IN(Contribution)
:    :         :····  OK
:    ⊟··· Waitingscreen
⊟··· Profit Display =|= (30)
     ⊟··· subjects.do {
          :····    SumC = sum(same(Group),Contribution);
          :····    N=count(same(Group));
          :····    Profit=Endowment-Contribution+EfficiencyFactor*SumC/N;
          :····    }
     ⊟··· Active Screen
     :    ⊟··· Standard
     :         :····  Your contribution to the project:  OUT(Contribution)
     :         :····  Sum of all contributions:  OUT(SumC)
     :         :····  Your Income in this period:  OUT(Profit)
     :         :····  continue
     ⊟··· Waitingscreen
```

To get z-Tree. . .

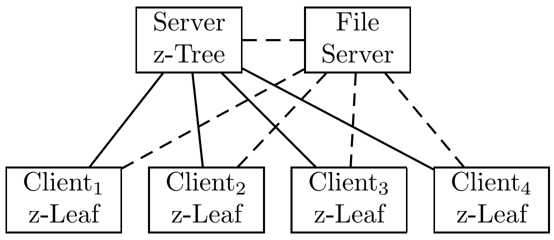
`http://www.kirchkamp.de/Cieyosh3/`
User: `imprs`
Password: `eox8dooJ`

1 Introduction

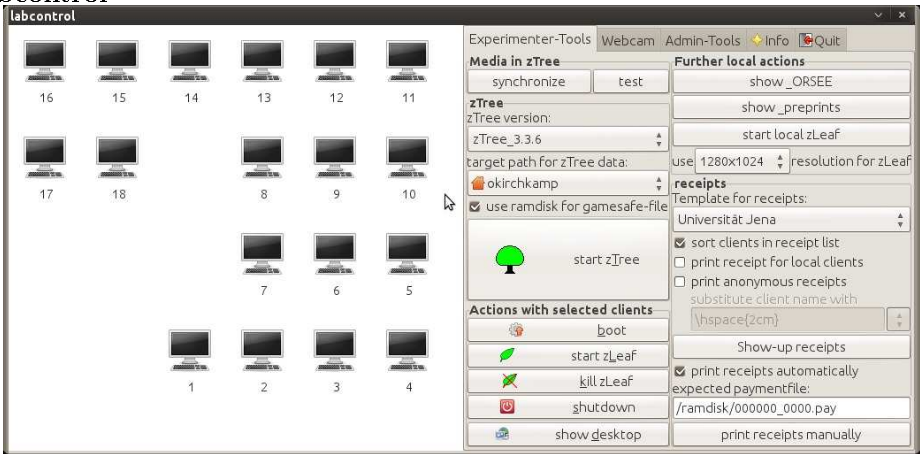
Basic concepts

- There is a server program `z-Tree` and a client program `z-Leaf`.
- In z-Tree experiments are **defined** and experimental sessions are **conducted**.

z-Tree and z-Leaf



Labcontrol



zTree

zTree version: zTree_3.3.6

target path for zTree data: okirchkamp

☑ use ramdisk for gamesafe-file

use 1280x1024 resolution for zLeaf

receipts

Template for receipts: Universität Jena

☑ sort clients in receipt list

☐ print receipt for local clients

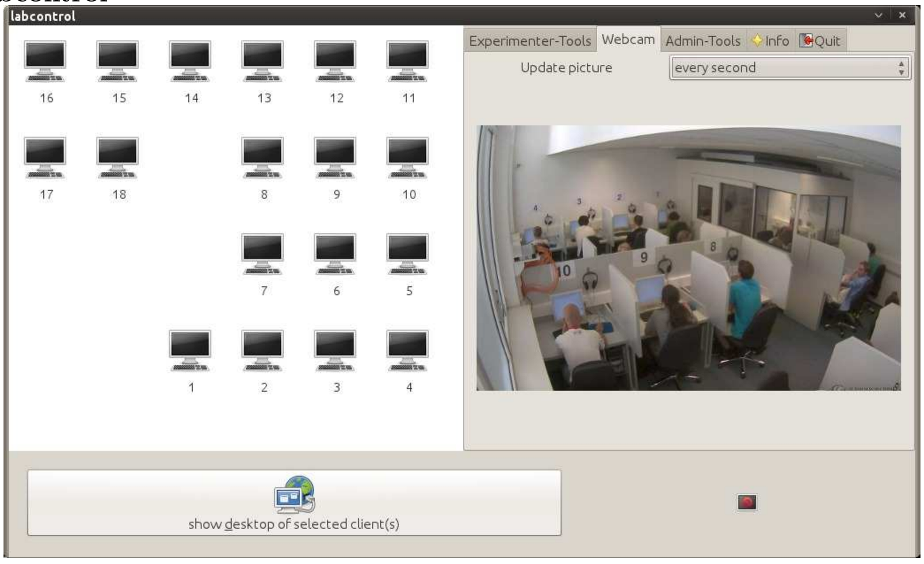
☐ print anonymous receipts

substitute client name with \hspace{2cm}

☑ print receipts automatically

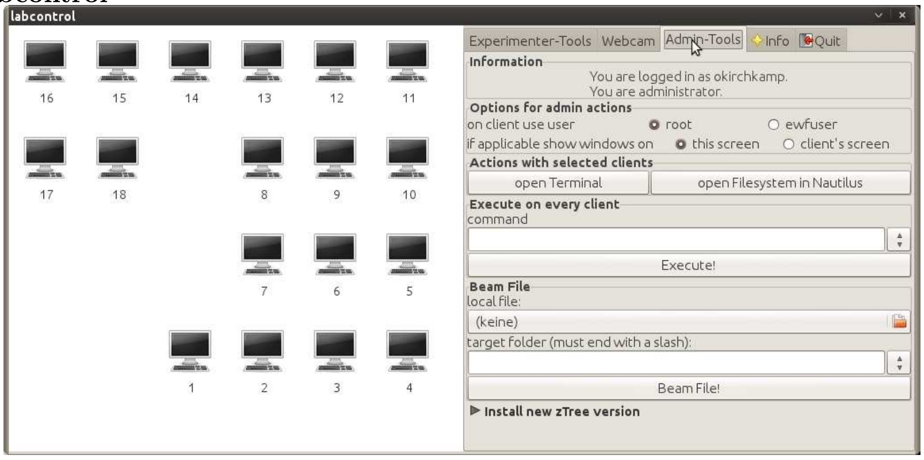
expected paymentfile: /ramdisk/000000_0000.pay

Labcontrol



Update picture every second

Labcontrol



You are logged in as okirchkamp.
You are administrator.

on client use user [✓ root] ewfuser

if applicable show windows on [✓ this screen] client's screen

A first step

- Start z-Tree
  - Change language
- Start z-Leaf
- Use alt - tab to switch from program to program.

Starting zLeaf

Linux — with a short script:

```
#!/bin/bash
num=2
for ((i=0;$i<$num;i++));
/usr/bin/wine zLeaf.exe /size 800x600 /name $i &
done
```

Microsoft — with links:

```
C:/.../zLeaf.exe /size 800x600 /name 01
C:/.../zLeaf.exe /size 795x605 /name 02
C:/.../zLeaf.exe /size 790x610 /name 03
C:/.../zLeaf.exe /size 785x615 /name 04
C:/.../zLeaf.exe /size 780x620 /name 05
```

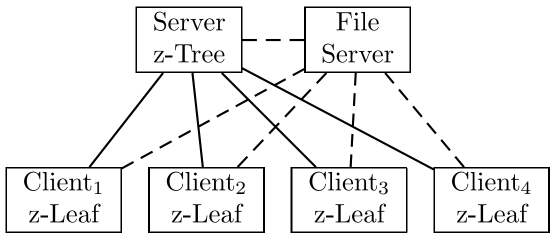
Stopping zLeaf

Linux — with a short script:

```
#/bin/bash
killall zLeaf.exe
```

Microsoft — using the task manager

Realtime z-Tree / Testing z-Tree

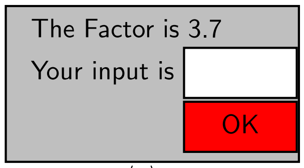


2 Individual decision making

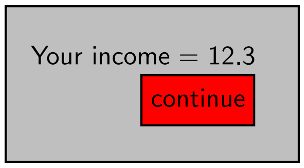
Exp1: The simplest game

- The subjects get to know a random number `Factor` between 1 and 3.
- The subjects enters a number between 0 and 5.
- The payoff is `Factor` times the number.
- What do the screens contain?
- How do we call the variables that are visible?
- What do we have to calculate?

```
Factor = 1+rounddown(random()*3,.1)
```



`Profit` $= \langle x \rangle \cdot$ `Factor`



Tables and Programming

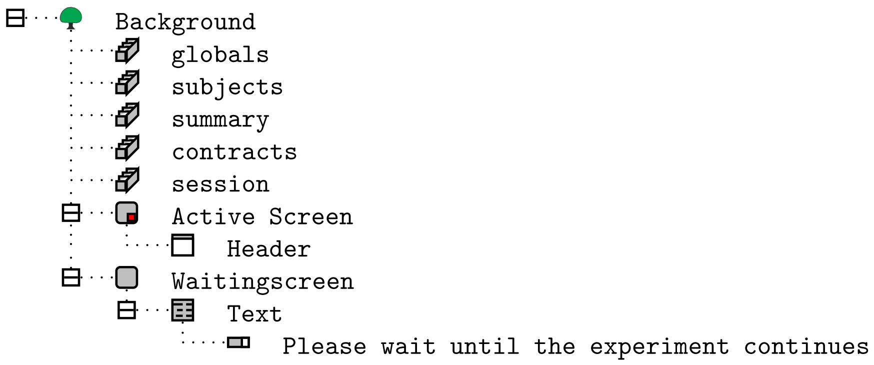
globals table			
Period	NumPeriods	RepeatTreatment	⋯
1	12	0	⋯

subjects table

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

- The data is stored in tables.
- The tables can be viewed in a window in z-Tree (menu Run/. . .)
- Rows are called records.
- Columns are called variables.
- Variables have names.
- Programs are executed in a record (in a table).

Tables and programming



```
Background
    globals
    subjects
    summary
    contracts
    session
    Active Screen
        Header
    Waitingscreen
        Text
            Please wait until the experiment continues
```

The globals table

globals table

Period	NumPeriods	RepeatTreatment	···
1	12	0	···

- Always only one single record
- One entry in the table for each variable.
- For each period there is a new ‘globals’ table
- assigning values to variables:

```
globals.do {
          variable = expression ;
          }
```

- operators and functions

```
globals.do {
          a = 12-3;
          b = 3;
          c = exp(b) - a * random();
          }
```

The subjects table

subjects table

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

- One row per subject.
- When a subject enters a stage, all programs at the beginning of this stage are executed.
- For each subject, programs are executed for the record in the 'subjects' database that belongs to the subject.
- For each period, there is a new 'subjects' table.

More on tables

	records	lifetime	execution	`variables`
`globals`	1	period	first subject	`RepeatTreatment`
`subjects`	subject	period	each subject	`Group, Profit, TotalProfit, Participate, LeaveStage, AuctionStop, AuctionNoStop`
`summary`	period	treatment	last subject	
`session`	subject	session	each subject	`FinalProfit, ShowUpFee,...`
`contracts`	dynamic	period	event	
`OLD...`				

Programming in the subjects table

```
Background
    globals
    subjects
    summary
    contracts
    session
    subjects.do {
        Factor = 1+rounddown(random()*3,.1);
        Endowment = 20;
        }
    Active Screen
        Header
    Waitingscreen
        Text
            Please wait until the experiment continues
```

Programming in the subjects table

```
    subjects.do {
        Factor = 1+rounddown(random()*3,.1);
        Endowment = 20;
        }
```

subjects table

Period	Subject	Group	Profit	TotalProfit	Participate	Factor	Endowment	···
1	1	1	0	0	1	3.2	20	···
1	2	1	0	0	1	2.1	20	···
1	3	1	0	0	1	2.6	20	···
1	4	2	0	0	1	2.3	20	···

Showing variables

```
Background
    globals
    subjects
    summary
    contracts
    session
    subjects.do {
        Factor = 1+rounddown(random()*3,.1);
        Endowment = 20;
        }
    Active Screen
        Header
    Waitingscreen
        Text
            Please wait until the experiment continues
Contribution Entry =|= (30)
    Active Screen
    Waitingscreen
```

Showing variables

```
Background
    globals
    subjects
    summary
    contracts
    session
    subjects.do {
            Factor = 1+rounddown(random()*3,.1);
            Endowment = 20;
            }
    Active Screen
        Header
    Waitingscreen
        Text
            Please wait until the experiment continues
Contribution Entry =|= (30)
    Active Screen
        Standard
            Your endowment:  OUT(Endowment)
    Waitingscreen
```

Data Input and Output

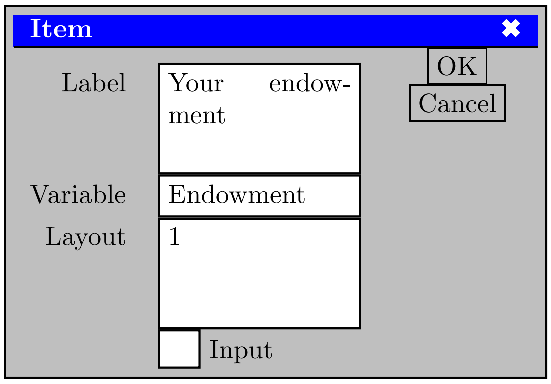
```
Contribution Entry =|= (30)
    Active Screen
        Standard
            Your endowment:  OUT(Endowment)
    Waitingscreen
```

- "Items" ▭ are used to display and to read in data.
- Items have a label that is displayed and used in error messages.

- Items are linked to a variable in a tables.

Showing variables

```
    Standard
        Your endowment:  OUT(Endowment)
```



Label: Your endowment

Variable: Endowment

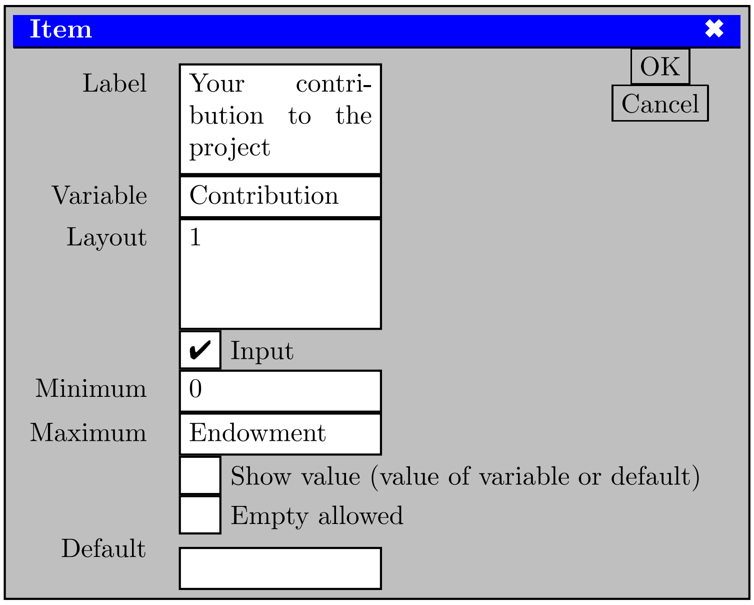
Layout: 1

- ... in the subjects table
- Items in default boxes
- Label
- Variable
- Layout

Showing variables

```
Background
    globals
    subjects
    summary
    contracts
    session
    subjects.do {
        Factor = 1+rounddown(random()*3,.1);
        Endowment = 20;
        }
    Active Screen
        Header
    Waitingscreen
        Text
            Please wait until the experiment continues
Contribution Entry =|= (30)
    Active Screen
        Standard
            Your endowment:  OUT(Endowment)
            Your contribution to the project:  IN(Contribution)
            OK
    Waitingscreen
```

Input of variables



Label: Your contribution to the project
Variable: Contribution
Layout: 1
✓ Input
Minimum: 0
Maximum: Endowment
Show value (value of variable or default)
Empty allowed
Default:

- ...into the subjects table
- Input items
- Automatic range check

Data Processing and Programming

- When subjects make input, the data is transferred to z-Tree.
- In z-Tree calculations are performed.
- The results of the calculations are sent to all the z-Leafs and displayed.
- Calculations can be executed (in programs) at the beginning of a stage and when buttons are clicked.

Concepts

- Subjects go through periods
- Periods are divided into stages
- Screens are composed of boxes
- Data is stored in z-Tree tables
- Programs allow us to modify the data (payoff functions)
- Data is shown in z-Leaf (in items) (or as plots)
- Data can be read in in z-Leaf (in items) (or as plots)
- Data is automatically saved

- Earnings are automatically accumulated

$$\texttt{TotalProfit} \leftarrow \sum_t \texttt{Profit}_t$$

Comments

- To make programs readable, insert comments
- There are two forms

```
// until the end of the line
/* multi ...
line ...
comment */
a /* comment within the line */ =2;
```

- comments cannot be nested

How to Run and Test z-Tree

- Start z-Tree on experimenter's computer
  - Change language
- Start z-Leaves on participant's computers
- ... start z-Leaf with a shortcut

  Create shortcut and add `/language english` in the target field.
- start z-Leaf with a batch file

  `zleaf.exe /language english`

How to build a test environment with several z-Leaves

- Put z-Tree and z-Leaf into one directory.
- Create shortcuts for z-Leaf with command line options or create a batch file

```
 zleaf.exe /name 1 /language english /size 640x480
 zleaf.exe /name 2 /language english /size 640x480 /position 10,10
```

-
  ```
  #!/bin/bash
  num=2
  for ((i=0;$i<$num;i++));
  /usr/bin/wine zLeaf.exe /size 800x600 /name $i &
  done
  ```

  (you can `move` windows under Linux, you can not move windows under Microsoft Windows.)

- Start z-Tree
- Start as many z-Leaves as necessary

Exercise: estimate.ztt

- Participant have to guess the value of a mathematical function.
- The closer they are, the higher their profit.

```
X = random()*pi()/2;
enter guess in an item
Diff = Guess - sin(X);
Profit = 1 - Diff * Diff;
// Profit is the variable that is relevant
    for payment
```

- Topics
  - Functions
  - Entry and display of variables

3 Game Theory

3.1 Symmetric normal form games

Exp 2: A public goods game

- Groups of 2; matching changes each period
- In each period each subject gets 20 points. These points can be kept or any amount can be invested in a public good.
- The profit of the subjects consists of two parts. First they get the points they kept. Second, the points in the public good are multiplied with 1.6 and divided equally between all members of the group.
- One point is worth .07 €.

Groups

- The variable Group determines the group matching.
- There are menu commands for different types of matchings.
  - Partner
  - Stranger
  - absolute Stranger
  - typed absolute Stranger
- You can modify the variable Group also in a program (recommended).

Tables and Programming

globals table

Period	NumPeriods	RepeatTreatment	· · ·
1	12	0	· · ·

subjects table

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

- The data is stored in tables.
- The tables can be viewed in a window in z-Tree (menu Run)
- Rows are called records.
- Columns are called variables.
- Variables have names.
- Programs are executed in a record (in a table).

The Stage Tree

The description of a treatment is arranged in a tree structure, the stage tree:

- The stage tree shows the sequence of stages
- Stages contain programs and the two screens ,
- Screens contain boxes
- Boxes contain items and buttons

In each stage the following happens:

- Can a subject enter stage?
- Programs are executed.
- Active screen is displayed.
- Waiting screen is displayed (if the next stage cannot be entered)

A Stage Tree

```
Background
    globals
    subjects
    summary
    contracts
    session
    subjects.do {
        EfficiencyFactor = 1.6;
        Endowment = 20;
        }
    Active Screen
        Header
    Waitingscreen
        Text
            Please wait until the experiment continues
Contribution Entry =|= (30)
    Active Screen
        Standard
            Your endowment:  OUT(Endowment)
            Your contribution to the project:  IN(Contribution)
            OK
    Waitingscreen
Profit Display =|= (30)
    subjects.do {
        SumC = sum(same(Group),Contribution);
        N=count(same(Group));
        Profit=Endowment-Contribution+EfficiencyFactor*SumC/N;
        }
    Active Screen
        Standard
            Your contribution to the project:  OUT(Contribution)
            Sum of all contributions:  OUT(SumC)
            Your Income in this period:  OUT(Profit)
            continue
    Waitingscreen
```

Table functions

- Sometimes we want to access data from other records
- Example

  `C` is the contribution to the public good

```
sumC = sum(C);
```

`sumC` is the sum of all entries in the column `C` of the subjects table.

- Syntax

```
table_function( expression );
table_function( condition, expression );
```

- $\text{sumC}_i = \sum_j (C_j)$

Other table functions are `average()`, `product()`, `count()`, `find()`, `maximum()`, `median()`, `minimum()`, `product()`, `regressionslope()`, `stddev()`

Scope operator

sometimes we want to mix data from the current record and from other records:

$\texttt{sumGH}_i = \sum_j (G_j \cdot H_i)$

```
sumGH = sum( G * :H);
```

Scope operator :

Examples

`C` is the contribution of a player.

`SumC` is the sum of all contributions in a group of players:

```
SumC = sum( Group == :Group, C);
SumC = sum( same(Group), C);

Rank = count( C >= :C );
```

Scope operator

```
c = sum( a * b);
d = sum( :a *b);
e = sum( :a * :b);
```

a	b	c=sum(a*b);	d=sum(:a*b);	e=sum(:a*:b);
2	5	10+48+56=114	10+24+14= 48	10+10+10=30
4	12	10+48+56=114	20+48+28= 96	48+48+48=144
8	7	10+48+56=114	40+96+56=192	56+56+56=168

same

subjects table						
Period	Subject	Group	Profit	TotalProfit	Participate	···
1	1	1	0	0	1	···
1	2	1	0	0	1	···
1	3	1	0	0	1	···
1	4	2	0	0	1	···

`same( x )` ⇔ `x == :x`

Examples

```
subjects.do {
        SumC = sum( same( Group ), C);
        AvOthC = average((same( Group )) &
            not(same(Subject )), C);
        }
```

Exercise rankesti

- The participants have to make a guess for a mathematical function.
- The profit depends on how good they are compared to the other participants.
- The treatment should work with groups.
- available table functions `average(); count(); find(); maximum(); median(); minimum(); product(); regressionslope(); sum();`

Solution rankesti

```
subjects.do {
        Diff = ...;
        }
```

Now we have to start a new program. All differences must be calculated before we can determine the rank.

```
subjects.do {
        Rank = count( Group == :Group &
            Diff <= :Diff);
        }
```

Other tables

- globals
  - one record for all subjects
- summary
  - one record per period
  - “survives” the end of the period
- contracts
  - used for market experiments
- session
  - similar to subjects table but “survives” the end of the treatment

Execute calculation in specific tables

```
subjects.do {
     a=1;
     }
```

- table.do

```
subjects.do {
     a=1;
     table.do {
         b=:a;
         }
     }
```

- table.tablefunction

```
subjects.do {
     myM = table.find(same(Group) &
                 not(same(Subject )), M );
     }
```

Scope and auto-scope

table A		
x	a	...

table B		
...	b	...

table C		
...	c	...

```
A.do {
     x = a + B.sum(:a + b + C.product(::a + :b + c))
     }
```

$$x_i = a_i + \sum_j \left(a_i + b_j + \prod_k (a_i + b_j + c_k) \right)$$

auto scope:

```
A.do {
     x = a + B.sum( a + b + C.product( a + b + c) )
     }
```

auto scope is confusing if the same name appears in different tables.

globals-DB

the globals table can be accessed with the scope operator as if the program was located in the globals table.

```
globals.do {
          subjects.do {
            myM = :M; }
          }
```

in the subjects table the following are equivalent

```
subjects.do {
          myM = :M;
          }
```

```
subjects.do {
          myM = \M;
          }
```

maximum scope = globals table

Exercise game222

- Make a treatment for a symmetric two person normal form game (e.g., a prisoner's dilemma) where both players have two strategies at their disposal.
- Subjects enter 1 or 2 to choose between the strategies.
- fine-tuning: try the input format:
  !radio: 1="A"; 2="B";

3.2 Asymmetry

Parameter table

- Parameters can be different for each period and for each subject:

  ```
  subjects.do {
            if ( Subject == 1 & Period == 1 ) {
                ...
                }
            }
  ```

- Or sometimes easier (?) . . . parameter table (in Menu)
- period parameters
- subject specific parameters

Exercise game222

- Generalize game222 to asymmetric games a treatment for a general two person normal form game where both player have two strategies at their disposal.
- Subjects enter 1 or 2.
- fine-tuning: use the input format:
  `!radio:  1="A"; 2="B";`
- The game parameters should change from period to period.

Exercise group

- Make treatments for pgsimple and game222 that can be conducted with more than one group.
- Try different group matchings:
  - Partner
  - Stranger
  - Matching groups
  - “absolute stranger”

3.3 Extensive form games

Examples

Public goods game:

all players:
1. contribution
2. profit display

Ultimatum game:

proposer	responder
1. proposal	[skips one stage]
[skips one stage]	2. acceptance
3. profit display	profit display

Leaving out stages

- Conditions
- Set the variable Participate to Zero;

```
subjects.do {
          Participate = if ( Type == PROPOSER, 1, 0);
          }
```

- alternatively:

```
subjects.do {
          if ( not ( Type == PROPOSER)) {
              Participate =0;
          }
```

Stage: start options

- Wait for all =|=
  - general case
- As soon as possible -=
  - simultaneous stages
  - sequence of stages that do not depend on other participants
- Start if. . . condition (. . .)|=
  - If condition is satisfied
  - Complex course of actions

Exercise: Ultimatum game

- Proposers choose offer $0 < x < 100$.
- Responders can accept or reject (Yes/No).
- If "Yes", payoffs are x for responder, 100-x for proposer
- If "No", payoffs are 0 for both

Item layout

layout	input	output
!radio: 1="Monday"; 7="Sunday";	○ Monday ⊙ Sunday	○ Monday ⊙ Sunday
!text: 1="green"; 2="red"; 3="black";	red	red

Exercise: Ultimatum game

```
globals.do {
     PROPOSER=0;
     RESPONDER=1; PIE=100;
     }
```

```
subjects.do {
     r = random();
     }
```

```
subjects.do {
     Type=if(r==maximum(same(Group),r),PROPOSER,RESPONDER);
     }
```

Proposal =|=

```
subjects.do {
     Participate=if(Type==PROPOSER,1,0);
     }
```

Standard
- You are a Proposer,...
- Piesize: OUT(Pie);
- Offer: IN(Offer);
- continue

Response =|=

```
subjects.do {
     Participate=if(Type==RESPONDER,1,0);
     }
```

```
subjects.do {
     Offer = find( same( Group ) & Type == PROPOSER,Offer);
     }
```

Standard
- You are a Responder,...
- Piesize: OUT(Pie);
- Offer: OUT(Offer);
- Accept: IN(Accept);
- continue

Exercise: Ultimatum game

Feedback Proposer =|=

```
subjects.do {
     Participate=if(Type==PROPOSER,1,0);
     }
```

```
subjects.do {
     Accept = find( same( Group ) & Type == RESPONDER,Accept);
     Share=if (Type=PROPOSER,Pie-Offer,Offer);
     Profit=Share * Accept;
     }
```

Standard
- You are a Proposer,...
- Piesize: OUT(Pie);
- Offer: OUT(Offer);

```
        Decision: OUT(Accept);
        Profit: OUT(Profit);
        continue
```

Feedback Responder -|=

```
    subjects.do {
        Participate=if(Type==RESPONDER,1,0);
        }

    Standard
        You are a Responder,...
        Piesize: OUT(Pie);
        Offer: OUT(Offer);
        Decision: OUT(Accept);
        Profit: OUT(Profit);
        continue
```

4 Layout

Screen layout

- Screen layout is static
- Screen is composed of boxes
- Boxes are placed sequentially

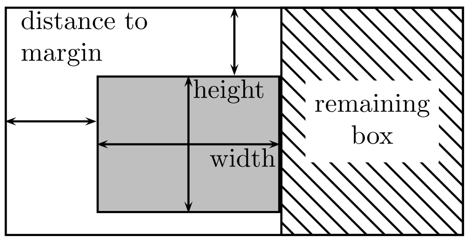
Box

Box = rectangular area of the screen containing stuff

- standard box
- header box
- help box
- grid box
- history box

Positioning of Boxes

- Boxes are positioned one after the other.
- There is a remaining box that can be cut.



Here the adjustment is at the *left* of the remaining box.

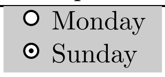
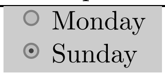
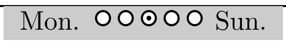
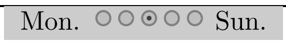
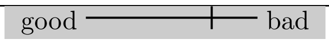
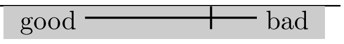
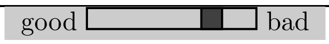
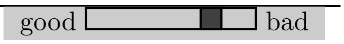
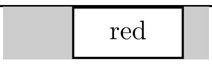
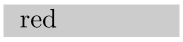
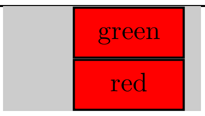
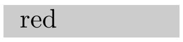
- Work from outside to inside.
- Use container boxes for parts of the screen that consist of several boxes, but logically belong together.
- Use container boxes when you need the same part of the screen in another stage.

Items

Item = informational unit

- number input
- text input (predefined text options)
- radio buttons
- check boxes
- sliders
- scrollbars
- buttons

Item layout options

layout	input	output
!radio: 1=”Monday”; 7=”Sunday”;		
!radioline: 0=”Mon.”; 6=”Sun.”;5;		
!slider: 10=”good”; 20=”bad”;101;		
!scrollbar: 10=”good”; 20=”bad”;101;		
!text: 1=”green”; 2=”red”; 3=”black”;		
!button: 1=”green”; 2=”red”;		

Active item

- When you put a program into a button, radio button,... the program is executed when the item is changed.

Exercise Layout

- Make radio buttons into your ultimatum game

5 Markets and Auctions

Sealed Bid Auctions

- simple

Clock Auctions

- later command:

```
globals.do {
          later (3) repeat { Price = Price -1;}
          }
```

- Leave the stage when one in the group accepted an offer in an auction. In a button

```
subjects.do {
          ...
          subjects.do {
              if ( same (Group) ) {LeaveStage = 1;}
              }
          }
```

- Exercise “Dutch auction”
  - Price starts at 100 and then decreases.
  - Each group has an inventory of 1 item which can be sold to a member of the group.
  - Once a member of a group has bought, all members move to the next stage.

Posted offer auctions

- Subjects can make an indefinite number of offers.
- Subjects see the other offers and can accept them.
- The stage is terminated after a fixed timeout.

Problems in market experiments:

- Not only “one” entry per stage.
- Participants get information on the entries of the other participants *during a stage.*
- Stage termination after timeout.

How to proceed?

- What do subjects see?
  - What is the contract data?
  - How do we represent it in tables?
  - Where do we put the information?
- What can subjects do?
  - How does data change when a subjects does an action (presses a button)?

$$\begin{tikzcd}
\textbf{z-Tree} & \textbf{z-Leaf} \\
\text{data} \arrow[r] & \text{display} \\
 & \text{action} \arrow[dl] \\
\text{program} \arrow[uu] &
\end{tikzcd}$$

“New” for market experiments

- contracts table.
  - table with flexible number of records:
  - records can be added
  - ... in contract creation boxes
  - ... with the new command:

```
contracts.new {
         x=1;
         }
```

- New types of Boxes:
  - contract creation box
  - contract list box
  - contract grid box
- Program execution triggered by button clicks.
- Automatic update of new and modified data.

contracts table in an auction
The contracts table contains all information about offers.

Price: The price offered

Seller: The ID (variable Subject) of the Seller or −1 if there is no seller, i.e., if the offer can be accepted by a seller.

Buyer: Analogous to seller ID.

Transactions:

- Bids: `Seller==-1`
- Asks: `Buyer==-1`
- Trades: `Seller==..., Buyer==...`

contracts table in an auction

Seller	Buyer	Price	Comment
2	−1	87	Seller 2 made an offer of 87
−1	6	17	Buyer 1 made an offer of 17
3	7	40	Buyer 7 bought from seller 3 at 40
1	−1	77	Seller 1 made an offer of 77
4	− 2	74	— deleted —
−1	9	33	Buyer 9 made an offer of 33
1	− 2	72	— deleted —
4	8	67	Seller 4 sold to buyer 8 at 67

- contract creation box — make contracts
- contract list box — view and select contracts
- contract grid box — view and modify a given number of contracts.

Stage termination

- Stage is left after a timeout…
- …or with the global variables:
  - `AuctionStop` (stop, even if more time is left)
  - `AuctionNoStop` (don't stop, even if time has run out)
- Single subjects leave a stage with
  - `LeaveStage`

Program in buttons
Example: Set the subject who created or selected the offer with:

```
      buy / ...
  subjects.do {
       CreatorOrSelector = :  Subject;
       }
```

- …executed when the button was successfully pressed.
- In these programs the record of subject who pressed the button can be accessed with the scope operator.

Contracts

```
Contract maker:  contracts
    Price:  IN(Price)
    Make offer
        contracts.do {
            Seller = :Subject ;
            Buyer = -1 ;
            }
Contract list:  contracts (Buyer==-1), sorted by:  Price
    Price:  OUT(Price)
    Buy
        contracts.do {
            Buyer = :Subject ;
            subjects.do { ;
                if(Subject==:Buyer) {
                Money = Money - Price;
                Assets = Assets + 1;
                }
                if(Subject==:Seller) {
                Money = Money + Price;
                Assets = Assets - 1;
                } }
```

Example: single sided auction

- contracts table:
  - Variables Seller and Buyer
  - Subjects ID : This subject made/selected the offer
  - −1: Open offer
  - −2: deleted offer
- Buyers make offers

  contract creation box:

  `Seller = -1; Buyer = :Subject; // scope operator`
- Sellers can accept offers

  contract list box

  `Seller = :  Subject;`

Exercise: single sided auction

- Show the own accepted contracts (auction2)
- delete the offers of the Buyer whose offer was accepted (auction3)

```
contracts.do{
   if ( Buyer == :Buyer & Seller == -1) {
     Seller = -2;
   }
}
```

Check entries with checkers

- checkers can be placed into buttons.
- Exercise
  - Add an improvement rule to the action: Only better offers are accepted.
  - Note: use `contracts.maximum(...)` not `maximum(...)`. The latter is executed in the new contracts only.

Examples

- Double auction (da)
  - Wage offer by firm or worker.
  - Acceptance by firm or worker.
  - Only one trade allowed.

Posted Offer Markets

- One proposer makes an offer. (e.g., in a contract creation box)

  ```
  Proposer = : Subject;
  Responder = -1;
  ```

- Responders can — one after each other — decide whether to accept or reject the offer.
- Option "At most one per group in stage"

  (If the acceptance decision consist of more than one stage: option ". . . and in previous stage(s)")

Posted Offer Markets

- Enter the stage if there are open offers

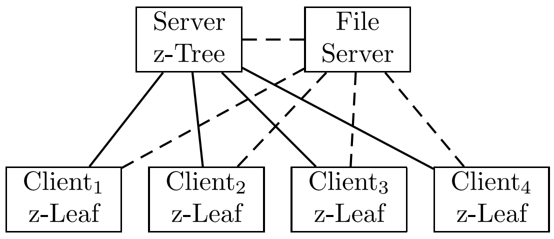
  ```
  if( Type == PROPOSERTYPE ) {
     Participate =0;
  }
  elsif( contracts.count(same(Group) & Responder > 0) >
     numOffers) {
        Participate =0;
  }
  else {
     Participate =1;
  }
  ```

Priority

- In stages with the "At most one per group in stage" option, the sequence of subjects can be determined with the variable Priority:
- The lower the value of Priority, the earlier a subject can enter the stage. (Think of the Priority variable as a rank.)

6 Running a session

z-Tree and z-Leaf



Planning a simple session

- welcome treatment (welcome.ztt)
  - setting the showup fee
  - calculator
- public goods experiment (pg.ztt)
  - the main treatment
- Ultimatumgame (ug.ztt)
  - a second treatment
- Questionnaires and payment (end.ztq)
  - payment file is written

6.1 Questionnaires

Questionnaires

- Payoff file can be written.
- Questions with no consequence on payoff.
- Different formats for the questions.
- Layout not screen oriented — indefinite end with scrollbar.
- Text entry possible.
- Some variables (`FinalProfit`,...) can be accessed.
- Importing questionnaires.

Typical Questionnaire

- address form (payment file)
- questionnaires
- profit display
- bye bye screen

Course of a session

- Preparation of treatments and questionnaires
- Start of experimenter PC. Start of z-Tree.
- Start of subject PCs. (automatic start of z-Leaf)
- Subjects show up
- Start of session with first treatment
- Observe subjects entries
- Start of further treatment(s)
- conclude session with questionnaire
- Payment
- Turn off computers
- Data analysis

6.2 Emergency handling

Crash of Subject PCs

- Subject PC can simply be restarted.
- If another computer is started as a replacement:
  - Discard client
  - Start replacement
  - Move the z-Leaf name in the clients' window onto the crashed computer's name.

Clients' Table		
4 clients	state	time
client 1		
client 2		
client 3		
reserve		

Crash of z-Tree

- Possibly reboot computer
- Start z-Tree
- Menu “Restart all Clients”
- Menu “Restore Client Order”
- Menu “Reload Database”
- Open last treatment
- Restart with negative number of practice periods

Losses and Bankruptcy

- Losses can be covered by
  - 1. Profit made in earlier periods
  - 2. Showupfee
  - 3. Credit or money added by the subjects
- How this is handled can be defined in the treatment.

Simple installation of z-Tree

- Make a directory “ztree”.
- Make a user “exp” that has read access to this directory.
- Make a batch file that starts zleaf.exe in the desired language.
- Put a shortcut of this batch file into the startup directory of the user exp.

Advanced installation of z-Tree

- Directory structure
  - ztree
    - datadir
    - priv
    - temp
    - testtrash
  - zleaf
- Make a batch file that starts zleaf.exe in the desired language and put a shortcut of this batch file into the startup directory of the user exp.
- Make suitable shortcuts to start ztree.exe

7 Advanced features

7.1 Graphics

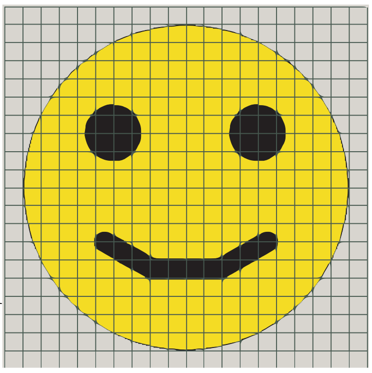
Graphics in Version 3

- Plot box
  defines a coordinate system for lineplots, scatterplots, bar and pie charts
- Plot point/line/rectangle/segment
  Graphics element with data in subjects table
- Plot text
- Plot graph
  Graphical display of (parts of) a table
- Plot input
  Click position translated into table entry
  Select, drag
- Multimedia box
  Display picture, sound or movie

Plot Box and Graphical Elements

- The plot box sets up a coordinate system.
  Linear, categorical, and logarithmic.
- Into such a coordinate system, graphical elements can be placed:
  Lines, arrows, circles, arcs.
- Positions defined in world coordinates (defined when the box is created)
- Line thickness… defined in screen pixels
- Graphical element have color defined in rgb(red,green,blue), r,g,b in [0,1].

Application: Smile



- Set coordinate system.
- Check “Maintain aspect ratio”.
- Add graphical elements.
- Use buttons to define color.
- Add a standard box with a button to conclude the stage.

Plot Graph

- Allows to draw graphs, charts,...
- Sets up a sequence of records
  - In each record, graphical elements can be drawn.
  - Points can be connected.
  - Plot graphs can also be nested.
  - Important: Only visible if it contains graphical elements!

Application: Grid lines

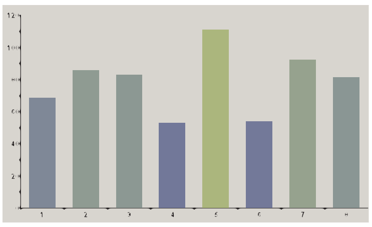
```
linepositions

globals.do {
     iterator(i,-100,100,10).do {
       linepositions.new{ pos = :i; }
     }

[-100,100]×[-100,100]
        ⋮
    graph:linepositions(TRUE)
        -100,pos->100,pos
    graph:linepositions(TRUE)
        pos,-100->pos,100
```

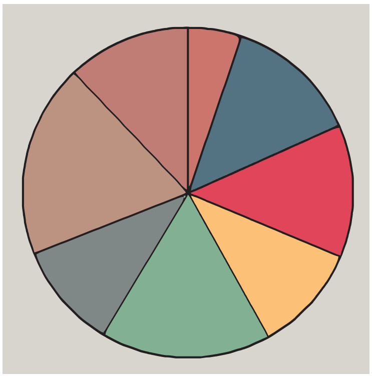
Application: Box chart

- Categorical coordinate system:
  `1..8 -> 0.5 ..  8.5`
- Plot graph with plot rect
- Axis with ticks and data labels.



Application: Pie chart

```
// Preparation
Sum = contracts.sum(Value);
Angle = 90;
contracts.do{
  Share = Value / :Sum;
  :Angle = :Angle - 360*Share;
  StartAngle = :Angle;
  Red = random();
  Green = random();
}
```



Application: Pie chart

Plot Pie ✖

Name	PIE	OK
center x	0	Cancel
y	0	
radius x	100	
y	100	
start angle[*]	StartAngle	
angle[*]	Share*360	
line color	rgb(0,0,0)	
line width	2	
fill color	rgb(Red,Green,.4)	

Graphical input

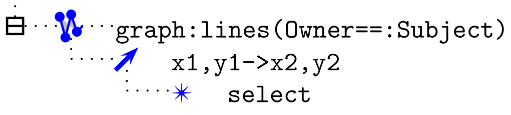
z-Tree	**z-Leaf**
data	display
	✳ click
program	

Example: Move a circle

- Initialize position (x,y) to (0,0)
- Define point at (x,y)
- Create plot input
  - Event: click; action: new
  - Directly input x and y into the subjects table.
    You loose information about clicking!
  - Perhaps better: Input into contracts table
    Add a program to modify data in subjects table.

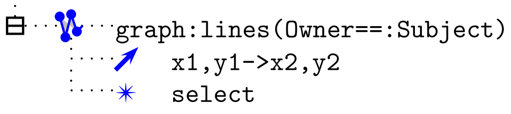
Graphical input: select

- Graphical objects can be selected as in contract selection box.
- Position of plot item determines which level is selected.
- Selects a line



```
graph:lines(Owner==:Subject)
    x1,y1->x2,y2
        select
```

- Selects the whole graph



```
graph:lines(Owner==:Subject)
    x1,y1->x2,y2
    select
```

- Selection can trigger a program for all the selected items, i.e., program is executed in scope of the selected object

Application: Highlight a rectangle when the mouse is moved into it.

- Event: MouseEnters/MouseLeaves
- Action: Select

Graphical input: drag

- The dragging checks whether an object has been clicked.
- While the mouse is clicked, the objects is dragged.
- While the object is moves, z-Leaf updates the object's position.
  - p0: where the objects has been clicked the first time
  - p: where the mouse is currently moved
  - p': where the object should be moved to
- When the object is released, z-Tree updates the object's position.

The multimedia box

- Box for displaying jpg pictures and movies.
- Must be located in the directory where z-Leaf runs.
- Cannot be places in plot boxes. (Is a to do for the developers).

Animation

- In combination with the later command, you can create animations.
- But be warned!
  - z-Tree is usually not efficient enough to create complex animations on the screen of each subject.

7.2 Chatting

Chat box

Chat box

- Input area like a contract creation box
- Output area like contract list box
- Display condition: will be box be displayed.
  `Type == Reader`
- Table: which table will be used (e.g. `contracts`).
- Input var: where messages are stored (e.g. `Words`)
  (if empty: observer, can only read, not write)

- Condition: which rows will be displayed.

  `Group == :Group`

- Output text:

  `<>Player <ID|1>: <Words|-1>`

- A program in the box allows to store additional variables:

  `Subject = :Subject; Group = :Group;`

7.3 Strings

Input items - strings

In a standard item:

- Layout: `!string`
- Input: ✓
- Minimum: (number of characters)
- Maximum: (number of characters)

Different from a chat:

- Return key does not send the message.

Strings

char(65)	"A"
code("A")	65
mid("haystack",4,3)	"sta"
pos("haystack","sta",1)	4
len("haystack")	8
upper("haystack")	"HAYSTACK"
lower("IMPRS")	"imprs"
trim(" hay stack ")	"hay stack"
format(3.1415926,0.01)	"3.14"
stringtonumber("3.14")	3.14

7.4 Data analysis

7.5 Experiment engineering

Experiment engineering

- Tips how to develop an experiment.
- Further programming concepts.

Tips how to develop an experiment

- What data will be presented in what order?
- How can this data be organized?
  - Think of different tables.
  - Use telling variable names.
  - Use comments.
- How is the experiment divided into treatments? (usually easy)
- What are the stages; the boxes?
  - You can simulate stages by sequentially showing different boxes.
- Design the screen layouts and describe the actions (programs).

Debugging

- Within programs
  - Read the error message.
  - z-Tree tries to position the cursor into the line where the error is located... look around.
  - Use comments to localize the error.
- Testing and debugging a treatment
  - Test each possible combination of parameters.
  - Consult the tables to check intermediate results.
  - Stop the clock.
  - Leave Stage (per client)
  - End after period
- Build an auto-pilot into your treatment. This facilitates testing the lab-performance with a larger number of clients.

Arrays

- Example: strategy method
  - Subject A selects a between 0 and 10
  - Subject B can condition b on a
- Solution 1:
  - `a, b0, b1, b2, b3,...`

    - `actualB = if( a==0,b0, if( a==1, b1, if ( a==2,....)))`

- Easier
    - array `b[0, 10];`
    - // enter a, b[0], b[1], b[2], b[3], ...
    - `actualB = b[a];`

Loops

- s = iterator(i, 5).sum(i*i);

  ```
  iterator( i, 0, 10 ).do{
     :b[i] = 10 + 2* i;// iterator opens new scope !!!
  }
  ```

- `while( condition )  statements ;`
- `repeat  statements  while ( condition );`

User defined tables

- Different markets
- Flexible history
- table definition
    - lifetime
        - period
        - treatment
        - session

Getting data from previous periods

- Use user defined tables.
- Use OLDsubjects, ... table.

Formatting with RTF

<u>RTF</u>

Text formatting in z-Tree

Example: If we want to write in **bold** or *italic*, we have to use RTF.

```
{{\rtf \fs18 {\ul\fs28\b\qc RTF\par }\b
Text formatting in z-Tree\b0 \par
Example: If we want to write in \b bold\b0  or
{\i italic}, we have to use RTF.}}
```

Variables integrated into text

Your income: 20 - 12 + 0.4 * 45 = 8 + 18.0 = 26.0
It is big.

```
<>Your income : <M|1> - <contribution|1> +
   <factor|0.1> * <sumC|1> = <Profit1|1> +
  <Profit2|0.1> = <Profit|0.1>
It is <Profit |!text: 0=‘‘small’’; 40=‘‘big’’;>.
```

Combining variables and RTF

- Variable are evaluated first.
- Conditional formatting is possible:
- Show the negative numbers in italic
- `<>{\rtf The number is <x|!text: 1="";-1="\i ">`
  `        <x|1><x|!text: 1="";-1="\i0 ">}`

Turning Boxes On and Off

- In all boxes there is an option

  display condition
- The variables used in this condition can change in the course of the experiment

  The box is hidden and shown
- By putting only a button an item into a standard box, you can also show and hide buttons or items

Menu: Leave Stage/End after period

- For debugging, a treatment can be ended prematurely.
  - At the end of a period.
  - Stages can be ended without subject intervention.
  - If input was necessary, a warning appears.
  - [ctrl]-[alt]-[F5] ends a repeat loop.
- However, no way to stop treatment.

Future Development

- Procedures and functions

Course of a session

- Preparation of treatments and questionnaires
- Start of experimenter PC. Start of z-Tree.
- Start of subject PCs. (automatic start of z-Leaf)
- Subjects show up
- Start of session with first treatment
- Observe subjects entries
- Start of further treatment(s)
- conclude session with questionnaire
- Payment
- Turn off computers
- Data analysis

Your treatment ready-made
Your treatment ready made:

- What decision have subjects to take ?
  variables
- When are decision taken?
  stages
- How do the screens look like ?
  Layout
- How choices interact and lead to payoffs ?
  Programs
- Does it work ? Debug

Operators

- Mathematical operators
  - + addition
  - - subtraction
  - * multiplication
  - / division

- Relational operators:
  - < smaller
  - <= smaller or equal
  - == equals
  - ! = unequal
  - >= greater or equal
  - > greater
- Logical operators:
  - & logical and
  - — logical or
  - Scope operators:
  - : next higher scope
  - \ highest possible scope. This is always the record of the globals table.

Statements

```
    x = y;
    if (a) ss1
    if (a) ss1 else ss2
    if (a) ss0 elsif (b1) ss1 elsif (b2) ss2...
    if (a) ss0 elsif (b1) ss1 elsif (b2) ss2....  else sse
    t.do ss
    t.new ss
    array v[ x ];
    array v[ x, y ];
    array v[ x, y, z ];
    while( a )  ss
    repeat  ss  while ( a );  later ( a ) do  ss
    later ( a ) repeat  ss
```

Functions

abs(x)	not(a)	average(x)
and(a,b)	or(a,b)	count()
atan(x)	pi()	find(x), find(a,x)
cos(x)	power(x,y)	maximum(x), maximum(a,x)
exp(x)	random()	median(x), median(a,x)
gettime()	randomgauss()	minimum(x), minimum(a,x)
if(a,x,y)	randompoisson(x)	product(x), product(a,x)
ln(x)	round(x,y)	regressionslope(x,y)
log(x)	rounddown(x,y)	regressionslope(a,x,y)
max(x,y)	roundup(x,y)	stddev(x), stddev(a,x)
min(x,y)	same(x)	sum(x), sum(a,x)
mod(x,y)	sin(x)	
	sqrt(x)	