

Part VI

Advanced concepts, questionnaires and crashes

Advanced concepts

Summary of z-Tree
tables

Questionnaires

Handling crashes

Advanced concepts, questionnaires and crashes

Advanced
concepts,
questionnaires and
crashes

Advanced concepts

Arrays

iterator

Data from previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree tables

Questionnaires

Handling crashes

Crash of a client PC

Crash of the server PC

Advanced concepts

Summary of z-Tree
tables

Questionnaires

Handling crashes

Arrays

Advanced
concepts,
questionnaires and
crashes

Arrays are *indexed variables*. In an index set, every finite, equidistant subset of numbers is allowed (e.g 1,2,3,...; but also 5, 10, 15, ...).

The index set needs to be defined *before* the first use of the array.

This is done with one of the following instructions:

- ▶ `array arrayname[]` → defines an array with indices from 1 to the number of subjects
- ▶ `array arrayname[n]` → defines an array with indices from 1 to n
- ▶ `array arrayname[x,y]` → defines an array with indices from x to y
- ▶ `array arrayname[x,y,z]` → defines an array with indices x, x+z, x+2z,..., y.

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

Arrays - an example

Advanced
concepts,
questionnaires and
crashes

```
//the index set of array p is 10, 15, 20  
array p[10, 20, 5];
```

```
p[10]=1;  
p[15]=5;  
p[20]=2;
```

```
p[30]=3; //sets p[20] to 3  
x = p[10] + p[20]; // x=4
```

arrayname[indexvalue] reports the element of array arrayname, corresponding to the index indexvalue.

Note: the expression indexvalue is rounded to the nearest possible index.

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

iterator

Advanced
concepts,
questionnaires and
crashes

Loops can be programmed with the statements `repeat` and `while`.

Another way of programming loops is by using iterators.

An iterator creates a *small temporary table* that contains only one variable.

When a table function or a `do`-statement is applied to an iterator, it corresponds to a **loop over the values contained in the table**.

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

iterator - example

This example provides two possible ways of calculating the sum of the square of the first five natural numbers:

$\text{SquareSum} = 1 + 4 + 9 + 16 + 25 = 55$

First way:

```
SquareSum = iterator(i, 1, 5).sum(power(i,2));
```

Second way:

```
SquareSum = 0;  
iterator(i,5).do{  
  :SquareSum = :SquareSum + power(i,2);  
}
```

[Advanced concepts](#)[Arrays](#)[iterator](#)[Data from
previous periods](#)[Bankruptcy rules](#)[Chats](#)[String variables](#)[Summary of z-Tree
tables](#)[Questionnaires](#)[Handling crashes](#)

iterator - example

This example provides two possible ways of calculating the sum of the square of the first five natural numbers:

`SquareSum= 1 + 4 + 9 + 16 + 25 = 55`

First way:

```
SquareSum = iterator(i, 1, 5).sum(power(i,2));
```

Second way:

```
SquareSum = 0;  
iterator(i,5).do{  
  :SquareSum = :SquareSum + power(i,2);  
}
```

Note the use of the **scope operator** to refer to a variable that is in the table *which contains* the table on which the program is running.

[Advanced concepts](#)[Arrays](#)[iterator](#)[Data from
previous periods](#)[Bankruptcy rules](#)[Chats](#)[String variables](#)[Summary of z-Tree
tables](#)[Questionnaires](#)[Handling crashes](#)

iterator: syntax

Advanced
concepts,
questionnaires and
crashes

An iterator has the following **syntax**:

- ▶ `iterator (varname)` → variable `varname` runs from 1 to the number of subjects.
- ▶ `iterator (varname, n)` → variable `varname` runs from 1 to `n`.
- ▶ `iterator (varname, x, y)` → variable `varname` runs from `x` to `y`.
- ▶ `iterator (varname, x, y, z)` → variable `varname` runs from `x` to `y` with steps of `z`.

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

Retrieving data from previous periods

In z-Tree, the subjects database is set up freshly in every period, so the information about earlier periods is not available directly.

However, the tables of the previous periods can be accessed with the prefix OLD.

So, if you want to copy some variable from the previous period, you can write:

```
if (Period >1) {  
  variable=OLDsubjects.find(same(subject),variable);  
}
```

This way, you can only access data from the previous period. To access data from earlier periods, a different procedure should be followed.

Advanced
concepts,
questionnaires and
crashes

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

Data from previous periods - example

Advanced
concepts,
questionnaires and
crashes

Suppose you want to run an experiment with the following structure:

1. N periods
2. in every period, each subject earns a certain amount of points
3. the subject's profit is determined at the end of period N:
 - ▶ three of the periods are drawn at random by the subject
 - ▶ the profit is set equal to the sum of the points earned by the subject in these three periods

Example: `values_from_previous_periods.ztt`

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

Advanced
concepts,
questionnaires and
crashes

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

click the button to select one period

draw a number

Data from previous periods - II

To retrieve data from all earlier periods, these data have to be stored in a **user-defined table** with lifetime set to Treatment or to Session.

From the menu, select Treatment→New table

Table

Name:

OK Cancel

Lifetime

☐ Period

☒ **Treatment**

☐ Session

Program execution

☐ When first subject enters stage

☒ Always when a subject enters stage

☐ When last subject enters stage

Advanced
concepts,
questionnaires and
crashes

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

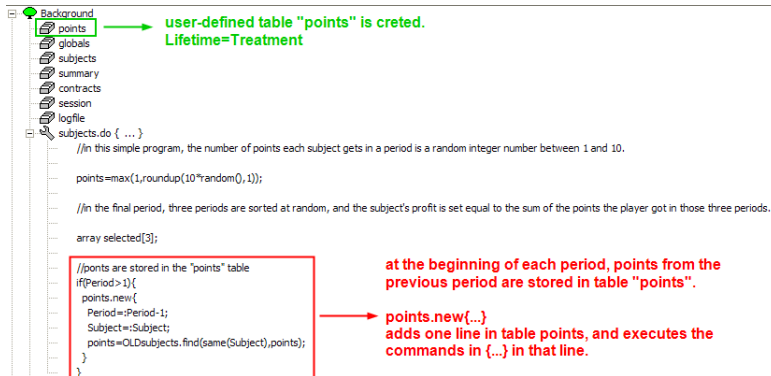
Summary of z-Tree
tables

Questionnaires

Handling crashes

Data from previous periods - III

Advanced concepts,
questionnaires and
crashes



The screenshot shows a z-Tree program editor. On the left, a file tree lists: Background, points, globals, subjects, summary, contracts, session, logfile, and subjects.do { ... }. The 'points' table is highlighted with a green box, and a green arrow points to a text box stating: "user-defined table 'points' is created. Lifetime=Treatment".

The main code area contains the following code:

```
//In this simple program, the number of points each subject gets in a period is a random integer number between 1 and 10.  
  
points=max(1,roundup(10*random(),1));  
  
//In the final period, three periods are sorted at random, and the subject's profit is set equal to the sum of the points the player got in those three periods.  
  
array selected[3];  
  
//points are stored in the "points" table  
if(Period>1){  
  points.new{  
    Period=:Period-1;  
    Subject=:Subject;  
    points=OLDSUBJECTS.find(same(Subject),points);  
  }  
}
```

A red box highlights the code block from `//points are stored in the "points" table` to the closing curly brace. A red arrow points from this box to a text box stating: "at the beginning of each period, points from the previous period are stored in table 'points'." and "points.new{...} adds one line in table points, and executes the commands in {...} in that line."

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

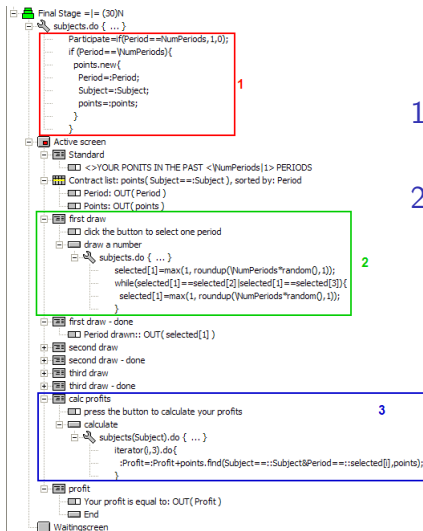
Questionnaires

Handling crashes

Note: again, here we use the scope operator to refer to variables in the subjects table from the points table

Data from previous periods - IV

Advanced concepts,
questionnaires and
crashes



1. store points form the last period
2. draw a random period
NB. this box is shown only if the draw is yet to be done.
Display condition:
`selected[1]==0`
3. calculate the profit, using the iterator.

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

Bankruptcy - I

Advanced
concepts,
questionnaires and
crashes

Tip: try to design the experiment in order to avoid possible losses.

If losses occur, can be covered by the following sources:

- ▶ profits from previous periods and treatments
- ▶ show-up fee
- ▶ money injected by the experimenter during the session

Example: bankruptcy.ztt

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

Bankruptcy - II

Advanced
concepts,
questionnaires and
crashes

If previous profits are not enough to cover all losses, but the **show-up fee** is, a message appears on the subject's screen.

This message informs the subject that he can choose *either to use the show-up fee, or to drop out of the experiment.*

- ▶ if the subject chooses to use the show-up fee, he may simply play on.
- ▶ otherwise, the subject reaches the state BankruptShowupNo (which you can read in the clients table), and the experimenter has to release him manually from the server.

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

Bankruptcy - III

If the **show-up fee cannot cover the losses** (or has already been exhausted) a message informs the subject that he has incurred a loss.

This message can be concluded with a question.

- ▶ By answering “no”, the subject enters the state BankruptMoreNo. In this case, the experimenter has to release the subject from the server.
- ▶ If the subject answers this question with “yes”, he enters the state BankruptMoreYes. In this case, there are 3 possibilities:
 1. Allow the subject to continue, by injecting into his account an amount of money higher than his current losses
 2. Another subject takes the role of the subject released
 3. The subject drops out

Advanced
concepts,
questionnaires and
crashes

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

Bankruptcy - IV

Advanced concepts,
questionnaires and
crashes

Bankruptcy Rules

General Parameters

Number of subjects: 1
Number of groups: 1
practice periods: 0
paying periods: 3
Exch. rate [Fr./ECU]: 0.1
Lump sum payment [ECU]: 20
Show up fee [Fr.]: 3

Bankruptcy rules...

Text "Do you invest your show up fee ?"
You have made losses. Would you like to invest your show-up fee to continue playing?

"yes" Yes Show up fee is invested and experiment goes on
"no" No Message "BankruptShowupNo" appears in client's table 1

Text "Do you want to go on"
You have made losses. Would you like to go on?

"yes" Yes Message "BankruptMoreYes" appears in client's table
"no" No Message "BankruptMoreNo" appears in client's table 2

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

* The experimenter can either
- give permission to go on (give a credit)
- replace the subject (nullify payoffs)

Cancel OK

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

Bankruptcy - V

Advanced
concepts,
questionnaires and
crashes

Clients' Table

	1 client	state	time
	2	BancruptMoreYes	-

Bankruptcy Continuation

Bankruptcy of subject 2/S 1

How to go on ?

☐ Subject can continue

☐ Other subject continues

Increase of credit limit (amount injected)

OK

session table

Subject	FinalProfit	ShowUpFee	ShowUpFeeInvested	MoneyAdded	MoneyToPay	MoneyEarned
1	-4.7000000000	3	1	3	1.29999999999999	-1.70000000000000

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

Chats

Advanced
concepts,
questionnaires and
crashes

Messages from the Left and Right-boxes.	Messages from the Left-box only.	
Sender 2, Left-box: a message from the left box Sender 3, Left-box: a message from the left box Sender 3, Right-box: an one from the right-box	Sender 2, Left-box: a message from the left box Sender 3, Left-box: a message from the left box	
	Messages from the Right-box only.	
	Sender 3, Right-box: an one from the right-box	
		my reply from the right-box

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Example: chat.ztt

Chat Box

Advanced concepts,
questionnaires and
crashes

In a stage, you can add a chat box by clicking on:
Treatment → New Box → New Chat Box

1. specify the table where you want to store the chat data contracts table user-defined table

Chat Box

Name: left: chat ☒ With frame

Width [p/]: Distance to the margin [p/]: 10%

Height [p/]: 0 Adjustment of the remaining box: 70%

☐ left ☐ top ☐ right ☐ bottom

Display condition:

1 Table: contracts

2 Condition: TRUE

3 Output text: <Sender <Owner{1}, <Box{text:1="Left"; 2="Right"}> box: <text{1}>

☒ Wrap text ☐ Output text centered

2. specify what records you want to display on the subject's screen
3. specify the output text

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

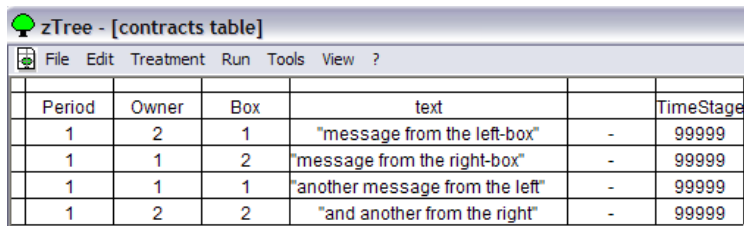
Summary of z-Tree
tables

Questionnaires

Handling crashes

Chat data

Advanced concepts,
questionnaires and
crashes



The screenshot shows a window titled "zTree - [contracts table]". It has a menu bar with "File", "Edit", "Treatment", "Run", "Tools", "View", and "?". Below the menu bar is a table with 7 columns: "Period", "Owner", "Box", "text", an empty column, and "TimeStage". The table contains 5 rows of data.

Period	Owner	Box	text		TimeStage
1	2	1	"message from the left-box"	-	99999
1	1	2	"message from the right-box"	-	99999
1	1	1	"another message from the left"	-	99999
1	2	2	"and another from the right"	-	99999

- ▶ messages are saved in the `contracts` table, or in the user-defined table specified in the Chat-box dialogue.
- ▶ The `contracts` table and the user-defined tables were the *only tables* in which text-variables could be saved, until version 3.4.
- ▶ To **display** text variables on the screen, write "-1" in the layout filed.

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

String variables

Advanced concepts,
questionnaires and
crashes

From z-Tree version 3.4 it is possible to enter text into input variables, using the layout `!string`.

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

- Label: (empty)
- Variable: word
- Layout: !string
- Input: ☒
- Minimum: 0
- Maximum: 9
- Show value (value of variable or default): ☐
- Empty allowed: ☒
- Default: (empty)

In **output items**, the expression in Layout should be an integer number, and determines how many characters will be reserved for the string output.

Example: string_variables.ztt

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

String functions

- ▶ `mid(str, n1, n2)` extracts $n2$ characters from string *str*, starting from character $n1$.
- ▶ `pos(str1, str2, n)` returns the position of *str2* in *str1*, starting the counting process at number n .
- ▶ `len(str)` returns the number of characters of string *str*.
- ▶ `upper(str)` returns string *str* in capitals.
- ▶ `lower(str)` returns string *str* in lower cases.
- ▶ `trim(str)` removes leading and trailing spaces from *str*.
- ▶ `format(n1,n2)` returns $n1$ as a string variable, rounded to the precision of $n2$.
- ▶ `if(a,str1,str2)` returns *str1* if a is true, otherwise *str2*.
- ▶ relational operators can be applied to pairs of sting expressions

Advanced
concepts,
questionnaires and
crashes

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Example

Advanced concepts,
questionnaires and
crashes

string_variables.ztt

- Background
 - globals
 - subjects
 - summary
 - contracts
 - session
 - letter_set**
 - letters**
 - words**
 - logfile
 - globals.do { //SET THE GLOBAL PARAMETERS ... }
 - subjects.do { //SELECT THE LETTERS IN EACH SUBJECT'S CHOICE SET ... }
 - letter_set.do { //SET THE LETTERS IN ALPHABETICAL ORDER ... }
 - Active screen
 - Waitingscreen
 - Text
 - Please, wait.
 - Anagrams =|= (60)A
 - Profits =|= (30)

Three user-defined tables. The first one contains the set of letters each subject can use, the second one the whole set of letters from which the first set is drawn, the third one the words typed in by the subjects.

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

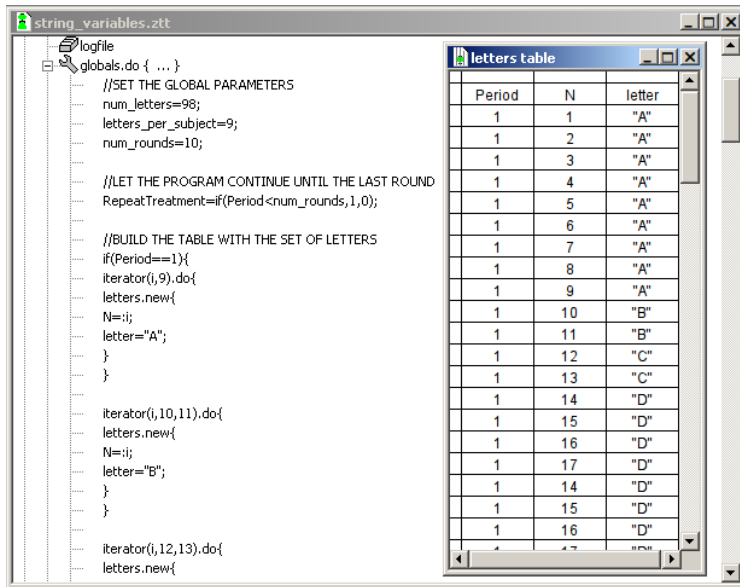
Summary of z-Tree
tables

Questionnaires

Handling crashes

Example

Advanced
concepts,
questionnaires and
crashes



The screenshot shows a z-Tree experiment interface. On the left is a script editor window titled "string_variables.ztt" containing a script. On the right is a data table window titled "letters table" displaying a table with 3 columns: "Period", "N", and "letter".

Script Editor Content:

```
logfile
globals.do { ... }

//SET THE GLOBAL PARAMETERS
num_letters=98;
letters_per_subject=9;
num_rounds=10;

//LET THE PROGRAM CONTINUE UNTIL THE LAST ROUND
RepeatTreatment=if(Period<num_rounds,1,0);

//BUILD THE TABLE WITH THE SET OF LETTERS
if(Period==1){
  iterator(i,9).do{
    letters.new{
      N=:i;
      letter="A";
    }
  }

  iterator(i,10,11).do{
    letters.new{
      N=:i;
      letter="B";
    }
  }

  iterator(i,12,13).do{
    letters.new{
```

Table Content:

Period	N	letter
1	1	"A"
1	2	"A"
1	3	"A"
1	4	"A"
1	5	"A"
1	6	"A"
1	7	"A"
1	8	"A"
1	9	"A"
1	10	"B"
1	11	"B"
1	12	"C"
1	13	"C"
1	14	"D"
1	15	"D"
1	16	"D"
1	17	"D"
1	14	"D"
1	15	"D"
1	16	"D"

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

Example

Advanced concepts,
questionnaires and
crashes

The screenshot shows a z-Tree experiment script named `string_variables.ztt` and a preview of the `letter_set` table.

Script Content:

```
session
letter_set
letters
words
logfile
globals.do { //SET THE GLOBAL PARAMETERS ... }
subjects.do { ... }
    //SELECT THE LETTERS IN EACH SUBJECT'S CHOICE SET
    repeat{
        letter_set.new{
            Subject=:Subject;
            repeat{
                rand=roundup(random()*num_letters,1);
                while(letter_set.count(same(Subject)&rand==:rand)>1);
            }
            while(letter_set.count(same(Subject))<letters_per_subject);
        }
    }
letter_set.do { ... }
    //SET THE LETTERS IN ALPHABETICAL ORDER
    letter=letters.find(N==:rand,letter);
    N=letter_set.count(same(Subject)&rand<=:rand);
```

letter_set table:

	Period	Subject	rand	letter	N
	1	1	27	"E"	3
	1	1	3	"A"	1
	1	1	6	"A"	2
	1	1	58	"N"	5
	1	1	40	"I"	4
	1	1	68	"P"	6
	1	1	94	"W"	8
	1	1	96	"Y"	9
	1	1	77	"S"	7
	1	2	8	"A"	2
	1	2	39	"I"	4
	1	2	54	"N"	5
	1	2	95	"X"	8
	1	2	94	"W"	7
	1	2	7	"A"	1
	1	2	97	"Y"	9
	1	2	26	"E"	3

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

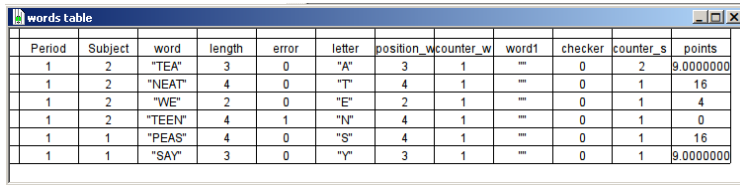
Handling crashes

Example

Advanced concepts,
questionnaires and
crashes

When a subjects types in a word, the program:

1. saves the word in the words table;
2. checks that the word contains only letters from the available choice set;
3. checks that it is the first time the subjects types in this word.



Period	Subject	word	length	error	letter	position_w	counter_w	word1	checker	counter_s	points
1	2	"TEA"	3	0	"A"	3	1	""	0	2	9.0000000
1	2	"NEAT"	4	0	"T"	4	1	""	0	1	16
1	2	"WE"	2	0	"E"	2	1	""	0	1	4
1	2	"TEEN"	4	1	"N"	4	1	""	0	1	0
1	1	"PEAS"	4	0	"S"	4	1	""	0	1	16
1	1	"SAY"	3	0	"Y"	3	1	""	0	1	9.0000000

The program **does not** check whether a word exists.
This should be done in case one really wanted to use this task.

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

Example



```
string_variables.ztt
├── Anagrams := (60)A
├── Active screen
├── Letters
│   ├── <> { \r\rf {f50 \qc \b\<letter_set.find(same(Subject)&N==1,letter)}1> \tab\<lett
│   └── Input
│       ├── { \r\rf {f50 \qc Please, form a word using the letters at your disposal.}
│       ├── IN( word )
│       └── OK
│           └── subjects.do { ... }
│               ├── words.new{
│               │   ├── Subject:=Subject;
│               │   ├── word:=upper(subjects.find(same(Subject),word));
│               │   ├── length:=len(word);
│               │   └── error:=0;
│               │
│               │   ├── iterator(i,length).do{
│               │   │   ├── letter:=mid(word,i,1);
│               │   │
│               │   │   ├── //count how many times the letter appears in the word
│               │   │   ├── position_w:=;
│               │   │   ├── counter_w:=1;
│               │   │   ├── word1:=mid(word,position_w+1,length-position_w);
│               │   │   ├── checker:=pos(word1,letter,1);
│               │   │
│               │   │   ├── while(:checker>0){
│               │   │   │   ├── position_w:=position_w+1:checker;
│               │   │   │   ├── counter_w:=counter_w+1;
│               │   │   │   ├── word1:=mid(word,position_w+1,length-position_w);
│               │   │   │   ├── checker:=pos(word1,letter,1);
│               │   │   │
│               │   │   │   └── }
│               │   │
│               │   │   ├── //count how many times the letter appears in the set
│               │   │   ├── counter_s:=letter_set.count(same(Subject)&letter==:letter);
│               │   │
│               │   │   ├── error:=if(:counter_w>:counter_s,1,error);
│               │   │   └── }
│               │
│               │   ├── error:=if(words.count(word==:word)>1,1,error);
│               │   ├── points:=power(length,2)*(1-error);
│               │   └── }
│               └── Contract list: words( same(Subject) )
```

Advanced
concepts,
questionnaires and
crashes

Advanced concepts

Arrays

iterator

Data from
previous periods

Bankruptcy rules

Chats

String variables

Summary of z-Tree
tables

Questionnaires

Handling crashes

The subjects table

Contains *one record per subject*. It is freshly set up for each period. The subjects table of the previous period is available under the name OLDsubjects.

In the subjects table, the following variables are always defined by z-Tree:

- ▶ Period
- ▶ Subject
- ▶ Group
- ▶ Profit
- ▶ TotalProfit - this is calculated automatically. It should not be changed manually.
- ▶ Participate
- ▶ LeaveStage

The globals table

Advanced
concepts,
questionnaires and
crashes

This table contains *a single record*, i.e. a single line. It stores values that are the same for all subjects. It is freshly set up for each period. The globals table of the previous period is available under the name OLDglobals.

In the globals table, the following variables are always defined by z-Tree:

- ▶ Period
- ▶ NumPeriods
- ▶ RepeatTreatment

Advanced concepts

Summary of z-Tree
tables

Questionnaires

Handling crashes

The summary and contracts tables

Advanced
concepts,
questionnaires and
crashes

The summary table

This table contains *one record per period*. It is not destroyed at the end of each period, as the subjects and globals table, but at the end of each treatment. This table is most useful for observing aggregate data of the treatment.

In the summary table, the only variable defined by z-Tree is the variable Period.

Advanced concepts

Summary of z-Tree
tables

Questionnaires

Handling crashes

The summary and contracts tables

Advanced
concepts,
questionnaires and
crashes

The summary table

This table contains *one record per period*. It is not destroyed at the end of each period, as the subjects and globals table, but at the end of each treatment. This table is most useful for observing aggregate data of the treatment.

In the summary table, the only variable defined by z-Tree is the variable Period.

The contracts table

This table is used mainly for market experiments and chats. New records can be added to this table, and existing records can be changed.

In the contracts table, the only variable defined by z-Tree is the variable Period.

Advanced concepts

Summary of z-Tree
tables

Questionnaires

Handling crashes

The session table

This table contains *one record per subject*. This table is never destroyed during a session, as it stores the aggregate profits earned by subjects in earlier treatments. It can also be used to *exchange information across treatments*, or *from treatments to questionnaires*.

In the subjects table, the following variables are always defined and automatically calculated by z-Tree:

- ▶ Subject
- ▶ Final Profit
- ▶ ShowUpFee
- ▶ ShowUpFeeInvested
- ▶ MoneyAdded
- ▶ MoneyToPay
- ▶ MoneyEarned

Questionnaires: Address form

Advanced concepts,
questionnaires and
crashes

Example: questionnaire_complete.ztq

The screenshot shows a window titled 'Adress' with a close button (X) in the top right corner. The form contains the following fields and controls:

- Adress Entry**: A text input field labeled 'Address'.
- First Name**: A text input field labeled 'First Name'.
- Last Name**: A text input field labeled 'Surname'.
- Adress**: A text input field labeled 'Street'.
- Postal code**: A text input field labeled 'Postal code'.
- City**: A text input field labeled 'Town'.
- Telephone**: A text input field labeled 'Telephone'.
- E-Mail**: A text input field labeled 'Email'.
- Do you want to participate in further experiments?**: A text input field labeled 'Would you like to participate in further experiments?'.
- Yes**: Two radio buttons labeled 'Yes' and 'No'.
- Continue (button label)**: A text input field labeled 'continue'.
- Help**: A text input field labeled 'Help'.
- Help text**: A text area labeled 'Please enter your name and adress. If your entries are confidential.' with a scroll bar.
- Buttons**: 'OK' and 'Cancel' buttons are located to the right of the 'Adress Entry' and 'First Name' fields.

The address form:

- ▶ usually goes first
- ▶ may/may not contain the name and the address of the subject
- ▶ as soon as it is filled in by all subjects, the **payment file** is generated by z-Tree.

Advanced concepts

Summary of z-Tree
tables

Questionnaires

Handling crashes

Question forms(s)

- ▶ contain several questions with different possible layouts.
- ▶ answers in questionnaires are of no consequences
⇒ no earnings can be made in questionnaires.
- ▶ answers are saved as text and cannot be used in programs.
- ▶ the last question form of a questionnaire remains on the users' screen until you continue (with a new treatment, a new questionnaire, or simply shutting down z-leaf).
- ▶ Therefore, the final question form *cannot contain a button*.

Questionnaire

Name (ID) Page 1

Title Page 1

Program Participate=if((Subject==1,1,0);

OK

Cancel

Layout of the questions

Advanced
concepts,
questionnaires and
crashes

Example of normal layout (not wide):

Text or number

radiobuttons ☐ A
☐ B
☐ C

radiolinelabel left right

radioline ☐ ☐ ☐ ☐ ☐ ☐ ☐

checkboxes ☐ M
☐ N
☐ O
☐ P

slider

scrollbar

buttons

Example of wide layout:

Text or number

radiobuttons ☐ A
☐ B
☐ C

left right

Left ☐ ☐ ☐ ☐ ☐ ☐ ☐ Right

checkboxes ☐ M
☐ N
☐ O
☐ P

slider L R

scrollbar L R

buttons

Advanced concepts

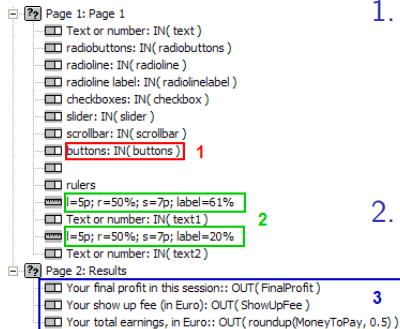
Summary of z-Tree
tables

Questionnaires

Handling crashes

Buttons, rulers, and variables

Advanced concepts,
questionnaires and
crashes



1. buttons: maximum one per question form.
⇒ clicking a button closes the question form, if possible
2. rulers: are used to set the regions where labels and questions are positioned.

3. In questionnaires, only variables from the session table can be retrieved. If you need to retrieve a specific variable from a treatment, you have to store it in the session table with a command in the treatment program (.ztt).

Advanced concepts

Summary of z-Tree
tables

Questionnaires

Handling crashes

Advanced
concepts,
questionnaires and
crashes

Crash of a client
PC

Crash of the
server PC

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

Crash of a client PC - II

If the first solution does not work, you can try to **move the subject to a different computer**.

1. Start a new PC and launch z-Leaf from there.
2. From the client table, **drag the name of the new client on the name of the client that does not work**. This will release the old client, and replace the old with the new client.

Crash of a client
PC

Crash of the
server PC

Clients' Table			
		state	time
	4 clients	*** Stage ***	-
	(1)	*** Stage ***	-
	2	*** Stage ***	-
	3	*** Stage ***	-
	4	*** Stage ***	-
	new_client		

Crash of the server PC - I

Advanced
concepts,
questionnaires and
crashes

After a **crash of z-Tree**, you have to follow this procedure carefully:

- ▶ restart z-tree
- ▶ open the client's table
- ▶ restart all clients with the menu Run → Restart all clients.

Advanced concepts

Summary of z-Tree
tables

Questionnaires

Handling crashes

Crash of a client
PC

Crash of the
server PC

Crash of the server PC - I

Advanced
concepts,
questionnaires and
crashes

After a **crash of z-Tree**, you have to follow this procedure carefully:

- ▶ restart z-tree
- ▶ open the client's table
- ▶ restart all clients with the menu Run → Restart all clients.
- ▶ If no clients connect you have different options
 - ▶ try to restart the clients manually
 - ▶ wait (up to 4 minutes) and try again
 - ▶ shut down and restart the experimenter's PC, then follow the previous steps
 - ▶ start z-Tree on a different computer

Advanced concepts

Summary of z-Tree
tables

Questionnaires

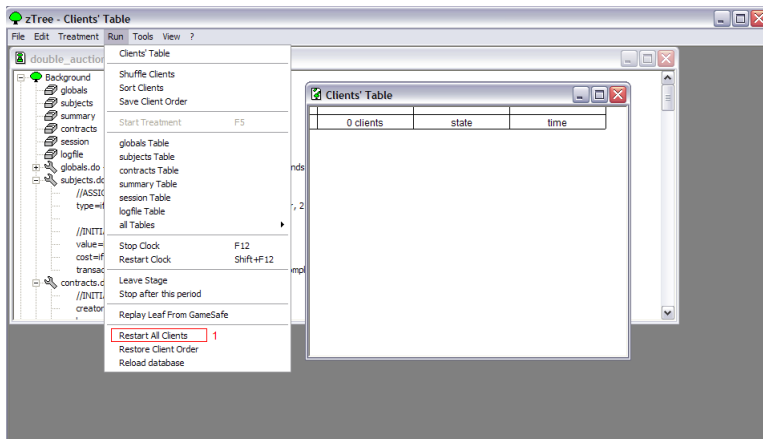
Handling crashes

Crash of a client
PC

Crash of the
server PC

Crash of the server PC - II

Advanced concepts,
questionnaires and
crashes



Advanced concepts

Summary of z-Tree
tables

Questionnaires

Handling crashes

Crash of a client
PC

Crash of the
server PC

Crash of the server PC - III

Advanced concepts,
questionnaires and
crashes

When you manage to restart all the clients,

- ▶ select Run → Restore Client Order (2).
- ▶ then Run → Reload Database (3).

Advanced concepts

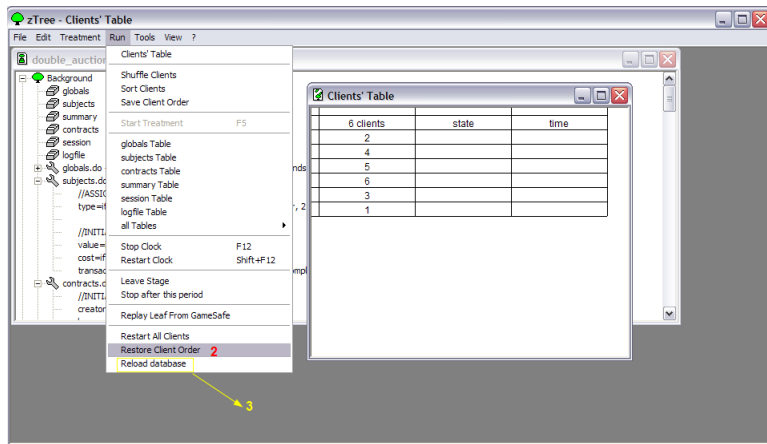
Summary of z-Tree
tables

Questionnaires

Handling crashes

Crash of a client
PC

Crash of the
server PC



Crash of the server PC - IV

Advanced
concepts,
questionnaires and
crashes

Advanced concepts

Summary of z-Tree
tables

Questionnaires

Handling crashes

Crash of a client
PC

Crash of the
server PC

- ▶ check **how many periods have been played** (e.g., from the subjects table)
- ▶ open the treatment that was running when the crash took place, and **set the number of practice periods to -n** (where n is the number of periods already played)
- ▶ select Run → start treatment.