

CESS: Unauthorized Z-Tree 3.0 Cheat-Sheet (Rough Draft)

- I. Commenting in Z-Tree: Methods & Comments on Style**
- II. Declaring Variables: Integer, Rational, and Arrays (Vectors)**
- III. Rounding Numbers**
- IV. Random Number Generation**
- V. Retrieving Values From Other Players, And Previous Rounds**
- VI. Flow Control (If Statements, and Loops)**
- VII. Managing Groups And Subjects**
- VIII. Graphical Interface For Z-Tree 2.X**
- VIII. Graphical Interface For Z-Tree 3.X**

I. Commenting Code In Z-Tree: A note on style:

There are generally two ways to write comments in Z-Tree:

“//”: All text preceding the “//” on the line where it appears is not interpreted.

A note on style:

In terms of readability, I use an individual program (subject.do, globals.do, etc) for each task that I would like to perform. Furthermore in the first line of each program I use the comment “//” followed by a brief description of what the program does, as shown *figure 1.1*.

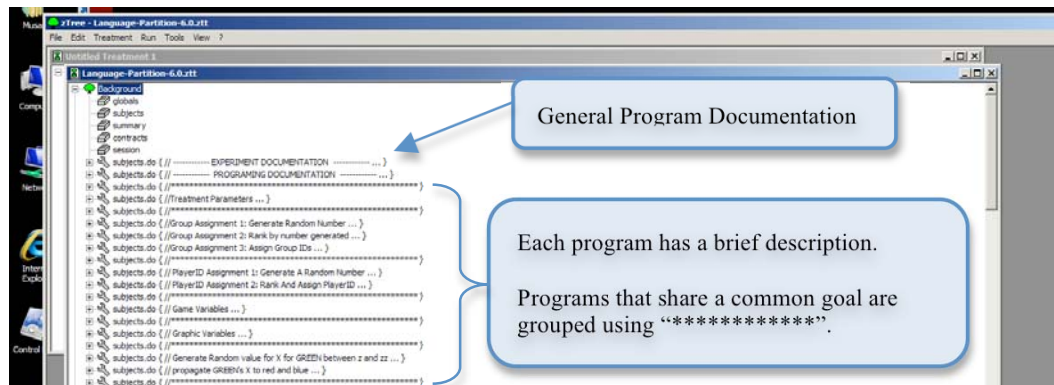


Figure 1.1

Within each program it is also a good idea to use not only descriptive names for your variables, as apposed to mathematical notation which is good for general description, but to also comment parameters as well. An example of this style of commenting is shown in *figure 1.2*.

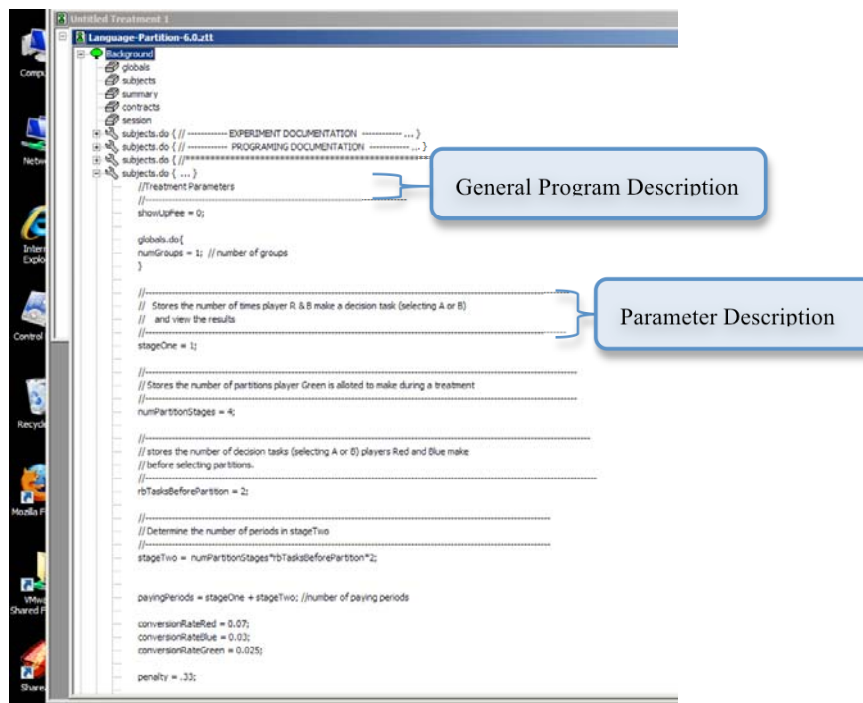


Figure 1.2

II. Declaring Variables: Integers, Arrays, and Matrices'

Declaring Integers:

variable = *integer*;

Example:

```
showUpFee = 10;           // assigns a 10 to variable show_up_fee
```

Declaring Rational Numbers:

variable = *rational_number*;

Example:

```
show_up_fee = 10.60;     // assigns 10.60 to variable show_up_fee
```

Declaring An Array:

```
array array_name[array_size];
```

The following declaration creates an *array_size* length vector, the first value (or cell) of which can be referenced using an index value of 1, whereby the last value can be accessed using an index value of *arraySize*. Note that the type "array" must precede the name you choose for the array immediately followed by an open bracket, the size of the array (vector), and the close bracket.

Example:

```
array earnings_each_round[3]; //creates a 3 element array
```

Assigning Values To Array Elements:

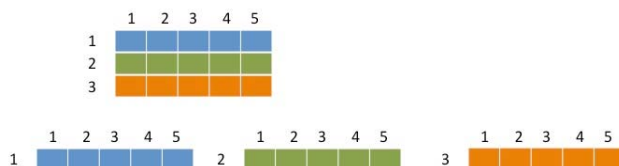
```
array_name[index_of_interest] = (integer value or rational_number);
```

Example:

```
earnings_each_round[1] = 10;
earnings_each_round[2] = 11;
earningsEachRound[3] = 12;
```

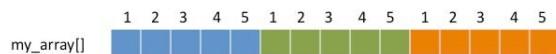
Pseudo Matrices'

Illustration of the conversion process from Matrix to Array



```
rows = 3;
columns = 5;
array_size = rows*columns;
```

```
array my_array[array_size];
```



The following index is used to access elements of the matrix by the number of columns, row, and column_number as follows:

```
index = columns*(row - 1) + column_number;
```

For Example:

```
// access row 3 column_number 4
```

```
index = 5*(3-1) + 4; // index = 14, which as per the true index in grey is correct
```

Description:

In order to convert a matrix into an array (*my_array*) it first must be declared, the size of which is equal to $array_size = rows * columns$. The elements in the matrix are accessed using an index = $columns * (row - 1) + column_number$.

Example 1:

```
row = 5;
column = 5;
array_size = row*column;
array my_array[array_size]; // create pseudo matrix
index = column*(2-1) + 1 // access row 2 column 1 using the index formula:
// index = column*(row-1)+col_num, variables are: row, and col_num;
```

III. Rounding Numbers

Using The Round Function:

```
variable = round(value_to_round, multiple_rounded_to );
```

The above rounds the number (*value_to_round*) to the nearest multiple (*multiple_rounded_to*).

Mental Illustration: The round function divides the number line starting from zero (in both directions) into partitions of length *multiple_rounded_to*. If the *value_to_round* lies anywhere at or above one of the midpoints between any two partition points it will be rounded up to the nearest partition point that lies above the *value_to_round*. If the *value_to_round* lies anywhere below the midpoint between any two partition points it will be rounded down to the nearest partition point below the *value_to_round*.

Example:

```
var = round(0.5, 1);           // rounds 0.5 to up to 1, which is the nearest partition point above 0.5, // and places it in  
var – since it is at the midpoint between two partition // points, specifically 0 and 1.
```

```
var = round(0.4, 1);           // round 0.4 down to 0, which is the nearest partition point below 0.4, // and places it in  
var – since it is below the midpoint between two // partition points, specifically 0  
and 1.
```

IV. Random Number Generation

Using The Random Function:

```
rand = random();
```

The random() function generates a random number between 0 and 1 inclusively.

Example:

```
rand = random();           // generates a random number between [0-1] and places it in rand.
```

Generating A Random Integer Between A Range (Inclusively):

```
rand = round((max_value - min_value)*rand(), 1) + min_value;
```

Example:

```
rand = round((7-2)*random(), 1) + 2;           //round((7-2)*random(), 1) first generates a random rational  
// number between [0,1] inclusively which is multiplied by  
// 5 = (7-2). The result of which produces a random number  
// between [0-5]. Adding two scales the random number  
// generated to a range of [2-7], the result of which is stored  
// in val.
```

Generating A Random Rational Number Between A Range (Inclusively):

```
value = (max_value - min_value)*random() + min_value;
```

Example:

```
rand = (10 - 5)*random() + 5;           //generates a random rational number between [5-10] which is stored rand.
```

Generating Random Player IDs Within Groups

The first program should contain the following:

```
rand = random();           // generate a random number between 0 and 1 inclusively
```

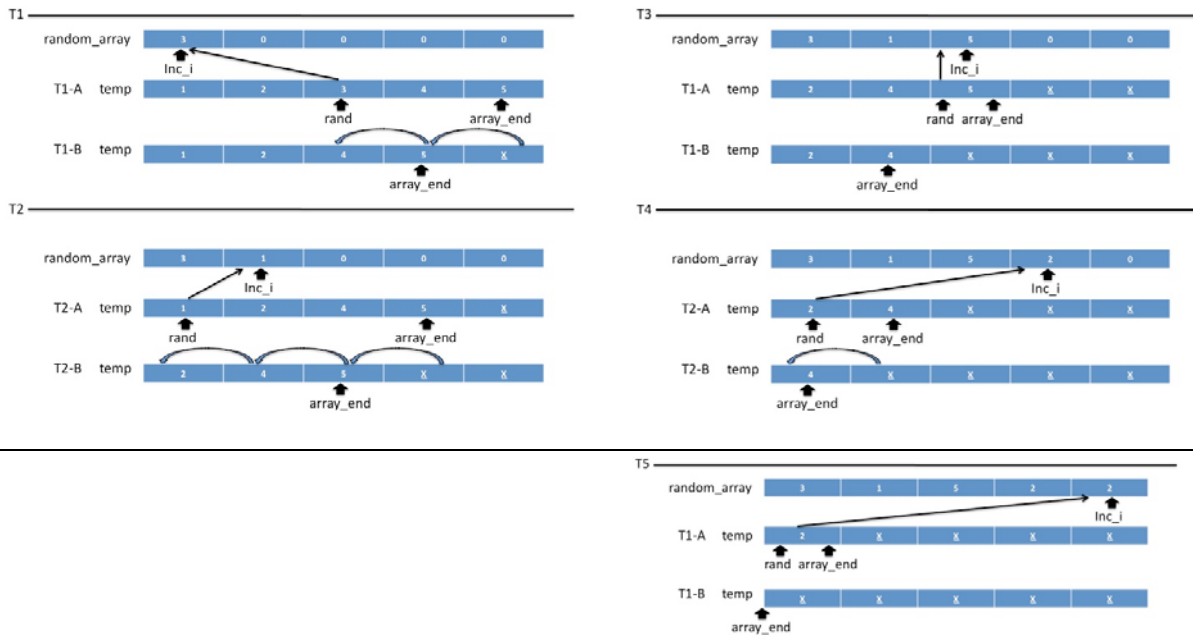
The second program should contain the following:

```
playerID = count( same(Group) & rand >=: rand); // rank the rand amongst group members, assign to playerID
```

Generating And Placing A Non-Repeating Random Series of Integers In An Array of Specified Length

Illustration of the algorithm:

The above algorithm essentially performs the following, where T1 represents the first iteration of the main while loop, T2 is the second iteration of the while loop, etc.



Description: Each subject creates a randomly permuted array from 1 to n, subjects table

The below sample code generates an array (for each subject) containing randomly permuted integers from 1 to *size_of_array*, which are stored in array *random_array*.

```
//Place In Subects.do
size_of_array = 10;      // size of array

array temp[size_of_array];      // temporary array
array random_array[size_of_array]; // array containing random permutation

inc_i = 1;
while(inc_i <= size_of_array){   // place values from 1 to size_of_array in the temporary array
    temp[inc_i] = inc_i;
    inc_i = inc_i + 1;
}

inc_i = 1;
array_end = size_of_array;

while(inc_i <= size_of_array){   // used place permuted values in random_array
    rand = round(random()*(array_end-1), 1) + 1; // generate a random number from 1 to array_end
    random_array[inc_i] = temp[rand];           // place randomly selected value in temp in random_array

    inc_k = if(rand != array_end, rand+1, rand); // inc_k contains the index to the right of temp[rand]
    while(inc_k <= array_end){                 // shift all elements to the right of temp[rand] left
        t = if(rand != array_end, inc_k-1, inc_k);
        temp[t] = temp[inc_k];
        inc_k = inc_k + 1;
    }
    array_end = array_end - 1; // decrement array_end since since there is one less number in temp[]
    inc_i = inc_i + 1;         // increment inc_i to the next element to insert in the random_array[]
}
}
```

Description: One subject creates a randomly permuted array from 1 to n, stored in the globals table

```
//Place in subjects table
globals.do{
size_of_array = 10;

array temp[size_of_array];
array random_array[size_of_array];
}

if(Subject == 1){

inc_i = 1;
while(inc_i <= size_of_array){           // place values from 1 to size_of_array in the temporary array
    temp[inc_i] = inc_i;
    inc_i = inc_i + 1;
}

inc_i = 1;
array_end = size_of_array;

while(inc_i <= size_of_array){           // used place permuted values in random_array
    rand = round(random()*(array_end-1), 1) + 1;    // generate a random number from 1 to array_end
    random_array[inc_i] = temp[rand];              // place randomly selected value in temp in random_array
    inc_k = if(rand != array_end, rand+1, rand);    // inc_k contains the index to the right of temp[rand]

    while(inc_k <= array_end){                   // shift all elements to the right of temp[rand] left
        t = if(rand != array_end, inc_k-1, inc_k);
        temp[t] = temp[inc_k];
        inc_k = inc_k + 1;
    }

    array_end = array_end - 1;                   // decrement array_end since since there is one less number in temp[]
    inc_i = inc_i + 1;                           // increment inc_i to the next element to insert in the random_array[]
}
}
```

V. Retrieving Values From Other Players, And Previous Rounds

Retrieving Values From Other Participants

```
variable = find(defining_condition, item_of_interest);
```

The find function searches for the *item_of_interest* belonging to the subject specified by a *defining_condition*.

Defining conditions can be specified using the same() function (e.g. same(Group), same(Subject), etc.) or logical conditions (e.g. Subject == 2, Group == 5, group_id ==6, etc).

Example 1:

```
current_var = find(same(Group), val);           // retrieve the value val from the subject with the same // group
                                                // value, which is stored in current_var
```

Example 2:

```
// The following retrieves the value stored in variable from subjects in the same group, and whom are
// of type green.
```

```
current_var = find ( same(Group) & playerID == green, variable);
```

Retrieving A Value In The Subjects Table From A Previous Round

```
variable = OLDsubjects.find(same(defining_condition), item_of_interest);
```

The method for finding the value of a variable from a previous round is similar to using the find() function, with the exception that the phrase "OLDsubjects." is prepended to find().

Example 1:

```
// retrieve the value var from a subject in the same group, and are type A
var = OLDsubjects.find(same(Group) & type == A , var);
```

Example 2:

```
//If in period 2 or greater, retrieve the subjects profit from the previous period
if( Period > 1) {
    subject_profit = OLDsubjects.find(same(Subject) , subject_profit);
}
```

Retrieving A Value In The Globals Table From A Previous Round

```
variable = OLDglobals.find(item_of_interest);
```

Example 1:

```
// Retrieve the previous number of players in each group from the globals table
group_size = OLDglobals.find(group_size);
```

Example 2:

```
// Retrieve the previous values of an array stored in the globals table
my_array[1] = OLDglobals.find(my_array[1]);
my_array[2] = OLDglobals.find(my_array[2]);
my_array[3] = OLDglobals.find(my_array[3]);
```

Note: the above lines of instructions cannot be placed in a while loop, as the first value retrieved from OLDglobals.find(my_array[1]) will be placed in my_array[1], my_array[2], my_array[3]. Hopefully this will be fixed in later versions of Z-Tree.

VI. Flow Control (If Statements, and Loops)

Using If Statements

```
if ( test_one ) {
```

```
    A
```

```
}
```

```
elseif ( test_two ) {
```

```
    B
```

```
}
```

```
else {
```

```
    C
```

```
}
```

```
D
```

// test_one is evaluated for a truth value (true/false) if it is true A is executed and moves to D if it is false test_two // is evaluated for a truth value. If test_two is true B is executed and moves to D, if it is false D is executed.

```
my_variable = if ( test, A, B );
```

If test is true my_variable is set to A, otherwise it is set to B.

Using While Loops

```
while ( test ){
```

```
    A
```

```
}
```

// While test is true A will be executed. Note: If a stopping case is not placed in A the while loop will execute indefinitely.

Using An Iterator to Loop Through Arrays

```
my_array[array_size];
iterator(array_index, starting_point, array_size).do{
    my_array[array_index] = some_value;
}
```

The iterator function loops through an array (*my_array*) using *array_index*, from *starting_point* to *array_size*, and places *some_value* in each corresponding element of the array.

VII. Stage Control

In order to control participation in a stage the variable *Participate* must be set to either 1 (participate in the stage) or 0 (do not participate in the stage) in a program placed in the aforementioned stage. Note: the variable *Participate* is set to 1 by default.

Example:

```
Participate = if (Round < 10, 1, 0);
```

The above states if the value of the Round variable is less than 10 then the player participates in the stage, otherwise the participant does not.

VIII. Programming: Managing Groups & Subjects

In order to set group matching using the Z-Tree build in matching feature select, treatment, Matching, and select one of the following options listed below:

Partner

The first subjects constitute group 1, the next, group 2, etc.

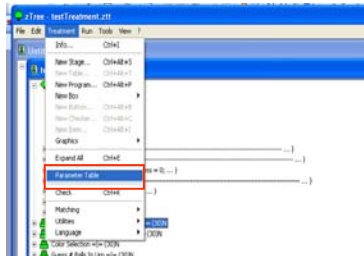
Example:

Subjects: 10

Groups: 2

The First set of 5 subjects will be in group 1, the second will be in group 2.

Note: Group matching can also be assigned to specified participants by selecting the appropriate region in the parameter table.



As First Period

Group matching set for the first period remains consistent throughout all other periods.

Stranger

Group matching is carried out at random for each period.

Absolute Stranger

Group matching is set such that each subject is never in the same group more than once.

Absolute Typed Stranger

This creates an absolute stranger matching for heterogeneous groups.

Retrieving Information From Previous Round

```
variable = OLDsubjects.find(same(Subject), variable);
```

Matching Participants

```
Group = rounddown( (Subject - 1) / numInGroup, 1) + 1;
```

Randomly Matching Participants

Create 4 individual programs. The first should be a globals.do program, and the following 3 should be subjects.do programs.

In the first program place:

```
numInGroup = some_number;          //(where some_number is an integer value greater than 1)
```

In the second program place the following line of code to generate a random number:

```
rand = random();
```

In the third program place the following to assign a unique random value to each subject:

```
RandomOrder = count(RandomNumber > :RandomNumber ) + 1;
```

Note: The above line of code can be read in English, from the perspective of a participant, as “My RandomOrder value equals the number of times another participant’s value for RandomNumber is strictly greater than my value for RandomNumber, plus one.” Adding one shifts the range [0, # of subjects - 1] to [1, # of subjects].

In the fourth program place the following line of code to assign a group ID to each participant:

```
Group = rounddown(( RandomOrder -1) / numInGroup, 1) + 1;
```

VIII. Graphical User Interface: Z-Tree version 2.5 and earlier

Item Display: Label

Displaying Variables in an Item box: (Z-Tree pre 3.0)

```
<>{\rtf <variable | 0.01> }
```

The 0.01 denotes the decimal place value that will be displayed

```
<>{\rtf <myArray[2] | 1.0>}
```

Displays the value stored in myArray at index 2

Conditional Display

```
<>{\rtf <variable | !text: 1="A"; 2="B"; 3="C";> }
```

Based upon the value of variable either A, B, or C will be displayed

Change Text Color

General Format:

```
{\rtf {\colortbl;\red0\green0\blue0;\redX\greenY\blueZ;} \cf2 enter text here \cf0 }
```

Where X, Y, and Z are integer values from 0 to 255.

Example:

```
{\rtf {\colortbl;\red0\green0\blue0;\red0\green255\blue0;} \cf2 enter text here \cf0 } // turns text green
```

Note: for more details on rtf color formatting see <http://en.wikipedia.org/wiki/Rtf>

Item Display: Layout

Radio Button

```
!radio: 1= "86.8"; 2= "102.8";
```



If the user selects 86.8 the value in the variable box is set to 1, If the user selects 102.8 the value in the variable box is set to 2.

Radio Line

```
!radioline: 0="zero"; 5="five"; 6;
```



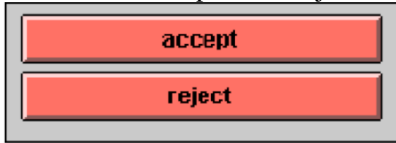
Check Box

```
!Checkbox: 1 = "check me";
```



Button

!button: 1 = "accept"; 0 = "reject";



X. Graphical User Interface: Z-Tree version 3.0 and above

Plot Box: Plot Text

Displaying variables in a plot text

<>< variable | 0.01>

Note: only one <> is needed at the beginning of a plot text

Example:

<> Player ID#<Subject | 1.0> // displays "PlayerID: 3" if Subject = 3

I. Images & Video Clips

In order for Z-Leaf to display images or video all video and image files must be placed on a shared drive that both the server (Z-Tree) and the client (Z-Leaf) can access.

In the following example the Z-Tree Server (**MyServer**) shares a directory called *Mapped* which contains all currently used Z-Tree images and videos. All of the Z-Leaf clients are mapped to shared directory *Mapped* on **MyServer**.

Note: The path pointing to the image must be specified, in this case, as Z:\image_name.jpg.