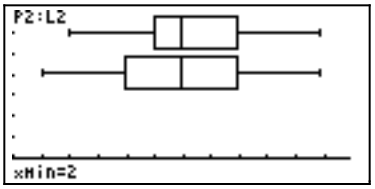
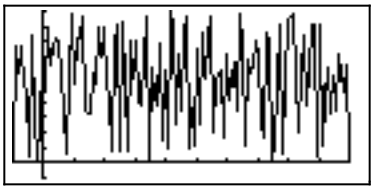
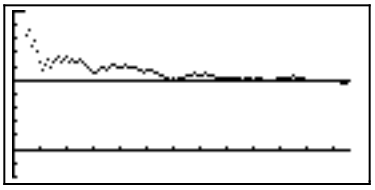
8. DEALING WITH RANDOMNESS







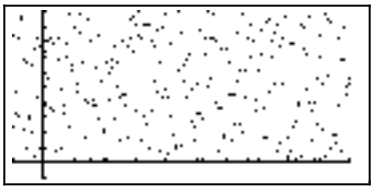


TABLE OF CONTENTS

Barry Kissane

The Australian Institute of Education
Murdoch University

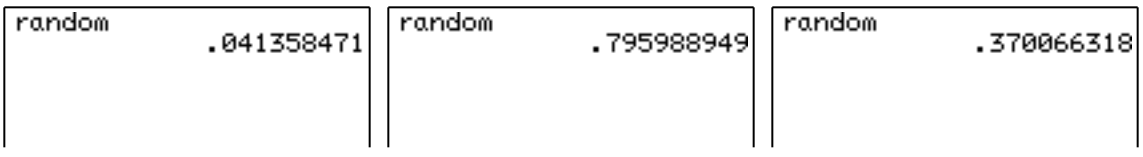
Many things that happen in the world seem to be predictable. For example, if you drop a coin, it is a pretty safe prediction that it will fall downwards rather than upwards. If you know enough about physics, you can even predict how long it will take to hit the ground, at least approximately, provided you know the height from which it was dropped. But there are many other things that are much less predictable. If the coin is spinning, you will not be able to predict confidently whether or not it will land with heads up or tails up, to give an obvious example. When something is unpredictable, it is often described as 'random'. This unit will allow you to explore some random events and to see how they can be studied using the Sharp EL-9600 graphics calculator.

Randomness is important in mathematics for at least three reasons. In the first place, some events are random in nature and cannot be understood well without understanding randomness. Many games are like this, including Backgammon and casino games. Secondly, many things that seem to be predictable are in fact random in nature. Some people have even suggested that nearly *everything* is random in nature if you look at it closely enough. (That's one of the reasons that you can't actually predict *exactly* how long it will take the coin to fall to the ground: there are too many random elements involved to do this.) Thirdly, some mathematical problems can be solved rather well using random procedures (called Monte Carlo procedures, named after the famous European gambling casino) even if they can't be solved in other ways.

8.1 Generating random numbers

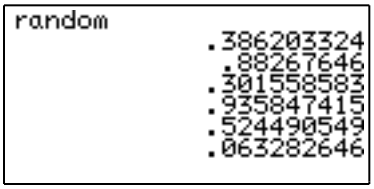
Your calculator has a command for generating numbers at random. Most of this unit relies on using this command. To paste the *random* command onto the calculator screen, press the MATH key, tap C to get the PROB (probability) menu and then tap 1 (or tap the *random* command twice) as shown below:

The *random* command will have no effect until the ENTER key is pressed, of course. When you press the ENTER key, the calculator will generate a number that is larger than zero and less than 1. Three possibilities are shown below, but you will probably get a different result from each of these.



If you press the ENTER key several times in a row, without entering a fresh command, the calculator will complete the previous command each time. So, to generate several random numbers, one after the other, press the ENTER key a few times.

The example below shows what happened when the key was pressed six times in succession.



Activity 1

1. Generate some random numbers. How often are large numbers generated? Small numbers? Try to describe the sizes of the numbers produced.

2. Sometimes, a random number has only eight digits (as in the last screen above). This is because the final digit was a zero and the calculator doesn't show it. About how often does this happen?

3 Examine the final digits of a succession of random numbers.
About how often is the final digit less than 4?
About how often is it an odd number?

4 Here is a game to play with a partner.
Generate a random number each.
The one with the larger number wins.
Play this game several times. Does one player seem to win more often than the other?
Repeat the game. How similar are the results?

The calculator's random number generator does not really produce numbers at random, of course, since it is a machine that has been programmed in advance. However, it produces numbers that *behave* as if they were randomly produced. (Sometimes, the numbers produced are called pseudo-random numbers.) Most importantly, the numbers produced will be *uniformly* (that is, evenly) spread out between 0 and 1 *in the long run*. So, about half of the time, the numbers produced will be less than 0.5. Similarly, about 60% of the numbers produced will be between 0.3 and 0.9.

Activity 2

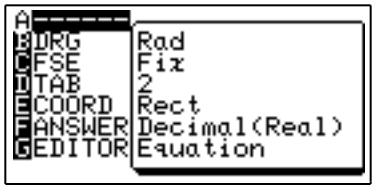
1. Generate a sequence of twenty random numbers by pressing the ENTER key twenty times in a row. (You can get seven random numbers – which is a screen full – by pressing ENTER seven times in a row.)
Note how many of the numbers were less than 0.5.

Repeat the experiment. Check with someone else's results.

2. Generate a large number (at least 30) random numbers.
Check to see that about 60% of the numbers generated are between 0.3 and 0.9.

3. You can generate numbers with less than nine decimal places by first changing the calculator SET UP. First press 2ndF BS (SET UP). The screen below on the left shows how to change the TAB to round results to two decimal places. The screen on the right shows how to set the calculator to show a fixed number of decimal places rather than a floating point number. (Press ENTER or tap again after selecting each setting.)

The new SET UP is shown below:



Now generate some random numbers and notice the effect of these changes.
Then try some other changes like these for yourself.

4. Set your calculator to show zero decimal places. (i.e. Fix 0).
Generate some random numbers.
Explain what happens.

Imagine you are tossing a coin, and let a zero refer to a tail and a one refer to a head. That is, let the number generated on the calculator represent the number of heads obtained when a coin is tossed (i.e., zero or one). Investigate how well the calculator models the tossing of a fair coin.

5. Set your calculator to Fix 0, as in activity 4.

Insert the *random* command and then press the ENTER key twice to simulate tossing a pair of coins. Are the two coins showing the same or different results?

Toss another pair of coins (by pressing ENTER twice more) and notice again whether or not the two coins show the same or different results.

Repeat this process many times, with a partner if possible.
Describe your results carefully.

6. Some people think that because there are three possible results when a pair of coins is tossed – two heads, two tails or one of each – that each of these three possibilities is likely to occur about as often as each of the others.

Use your calculator to check out this for yourself.

When you have finished Activity 2, press SET UP and change the calculator back to display Floating Point numbers for later work in this unit.

The idea of simulation is very important here. The word *simulate* comes from the Latin word *simul*, which means 'same'. Some common English words are derived from this stem, such as simultaneous (at the same time). We simulate something when we produce things that are the same in some important ways.

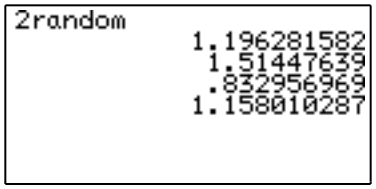
8.2 Transforming random numbers

For most practical purposes, random numbers between zero and one are not directly useful. Instead, we need random numbers of other kinds. The capacity of the EL-9600 to *transform* numbers is especially useful for doing this.

Consider, for example, the problem of generating random rolls of a standard die, like those used in many games. If we could use the calculator to generate (i.e. to simulate) the die rolls, we could easily study what happens in the long run with many die rolls.

Activity 3

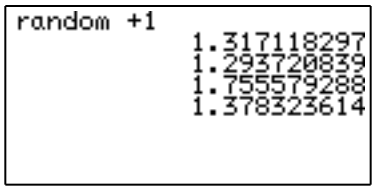
1. Investigate what happens when random numbers generated by the calculator are all doubled. To do this, enter the command as shown below and press the ENTER key several times.



   Do this with a partner until you are convinced that you know what kinds of results are obtained. What is the largest number you get? The smallest? Why?

   Write down your conclusions and explain your results.

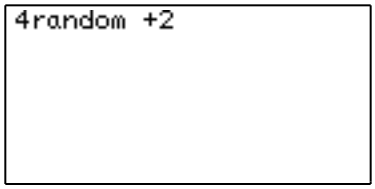
2. What kinds of numbers would you expect to get if you generated random numbers and then added one to each number? The screen below shows how to start doing this.



   Do this with a partner until you are convinced that you know what kinds of results are obtained. What is the largest number you get? The smallest? Why?

   Write down your conclusions and explain your results.

3. Predict what will happen if you press ENTER several times in succession after entering the command shown below. Tell your partner what you expect to happen *before you touch the calculator.*



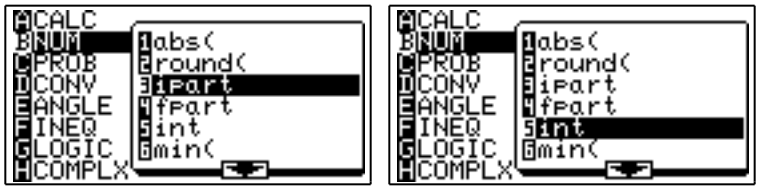
Test your prediction by actually generating numbers in this way. What is the largest number you get? The smallest? Why?

Write down your conclusions and explain your results.

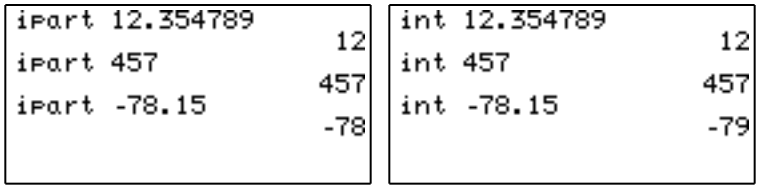
4. What command will generate random numbers between 1 and 7? Test your answer by using it a number of times until you are convinced that it produces (only) the desired results.

Activity 3 suggests how you can generate random numbers on an interval different from that from zero to one. To generate random dice rolls, however, we want integers (like 2, 4 and 5) rather than numbers with several decimal places to the right of the decimal point (like 2.234526162 and 4.762951324).

One way of obtaining random integers is to simply ignore everything to the right of the decimal point. That is to interpret 2.234526162 as a 2 and 4.762951324 as a 4. But there is a neater way of doing so, using the calculator's integer part commands, *ipart* or *int*, shown on the screens below. Notice that each command is available in the NUMerical part of the MATH menu.



The *ipart* command gives the integer part of a number – that is, the whole number part. The *int* command gives the greatest integer less than or equal to a number, and so in mathematics it is usually called the *greatest integer function*. The screens below show some examples of these two functions in use.



Activity 4

1. Experiment a bit with the *int* and *ipart* commands to see how they work. Make sure that you

2. Look carefully at the two screens above. Notice that the two commands do not always give the same results.

   For which numbers will the commands give the same results? For which numbers will they give different results?

   Explain why the results are not the same.

Calculator commands can be put together to get more complicated transformations than those you have seen so far. The command shown on the screen below, for example, will simulate rolling a standard six-sided die, for which each of the six faces is equally likely to be obtained.

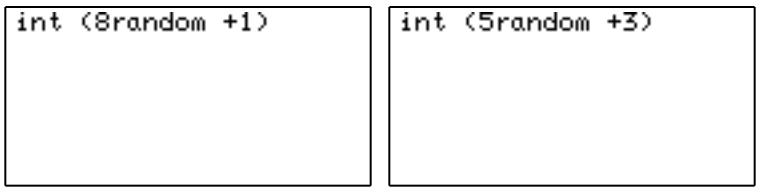
In this particular case, the calculator has simulated five rolls of a die, producing a 1, a 3, two 4's and then a six.

Activity 5

1. Use the command *int (6random +1)*, shown in the screen above, several times by entering it and repeatedly pressing the ENTER key. Make sure that the results are consistently integers between 1 and 6, and that all six possibilities occur.

2. What would you expect to happen if you used the *ipart* command instead of the *int* command in activity 1? Test your prediction by making the change and trying it out.

3. The screens below show two different transformations of the *random* command. For each transformation, predict what will happen if you generate a set of random numbers with the command. Discuss your prediction with your partner.



   When your are confident of your prediction, generate some numbers to test it.

4. What calculator command will generate random numbers to simulate the rolling of a 10-sided die, for which each of the faces showing 1, 2, 3, ... , 10 is equally likely to appear?

5. Explain how you could simulate birth months using the calculator, assuming that people are equally likely to be born in each of the twelve months of the year.

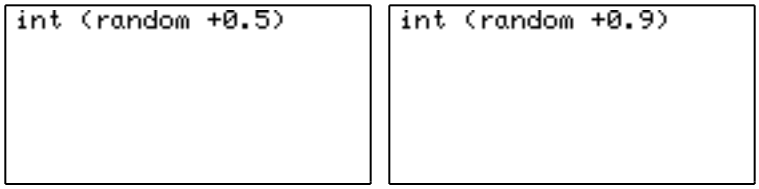
   (Incidentally, do you think this is a good assumption? Give at least two reasons for being

6. A class has 31 students in it, and you are asked to choose a student at random to win a prize. 'At random' means here that all students have the same chance of being chosen. Explain how you can use the calculator to do this task.

7. *Oz Lotto* is a legal gambling game in which adults buy a ticket consisting of six numbers chosen from the set of integers, {1, 2, 3, ... , 45}. Design a transformation of the *random* command to choose a set of six numbers for a ticket. Notice that the same number cannot be chosen twice for a ticket.

   Choose a set of six (different) numbers and compare it with the most recent results for *Oz Lotto*, which are probably published in a newspaper in your nearest capital city. Since almost everybody who buys a ticket in *Oz Lotto* loses their money, your simulated ticket will probably lose too. Check this for yourself.

8. Generate a number of random numbers using each of the following two commands.



   Explain carefully and as completely as you can what the differences between the two sets of numbers generated are. You will need to generate a large set of numbers in order to see the results thoroughly.

Commands like those used in question 8 of Activity 5 are especially useful. You will have noticed that they produce numbers that are either zero or one. These can be used to represent things that happened (1) or didn't happen (0). An example is tossing a coin. A result of one can represent a head and a result of zero can represent a tail. The result, 0 or 1, can also be used to mean the *number* of heads obtained (i.e. 0 or 1).

To see why the results are like this, consider carefully a command like *int* (*random* + 0.5).

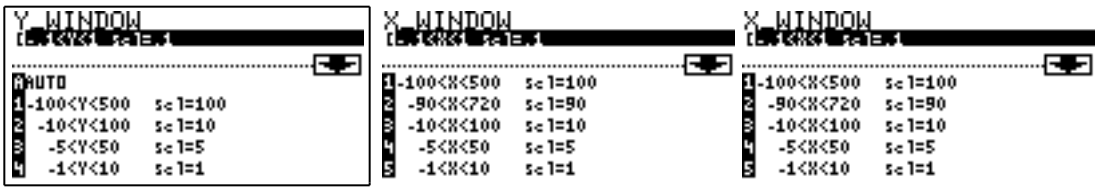
The *random* command produces numbers that are larger than zero and less than one. So the command *random* + 0.5 produces numbers that are larger than 0.5 and less than 1.5. When the *int* command is applied to the result, it will give a zero for numbers between 0.5 and 1 and a one for numbers from 1 to 1.5. If you think about this carefully, you can see that about half the time you will get a zero and about half the time a one. This is a good simulation of tossing a fair coin.

Importantly, each time the command *int* (*random* + 0.5) is used, the result does not depend on the previous result, which is also important for simulating the tossing of a fair coin. Some people describe this by saying, "the coin has no memory".

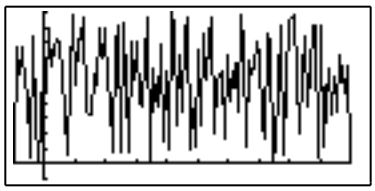
8.3 Displaying randomness

So far, you have generated random numbers one at a time. But randomness can be displayed on your calculator both graphically and numerically in larger amounts. This will give you some insight into how the calculator works, too.

To see a graphical display of randomness, simply graph the *random* function. Since the values of this function will be between zero and one, a good choice of window is the first quadrant. Use the WINDOW EZ choice 3 to set the calculator up as shown below.

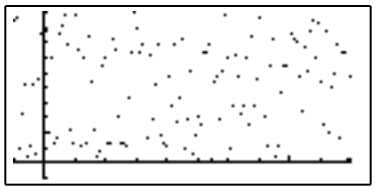


When you press GRAPH, the resulting image of randomness will look something like the one below, but will not be the same.



Notice that the calculator plots one point at a time and then connects it in a line to the previous point. You can get another image of randomness by plotting the points as dots on the screen but not connecting them together. To do this, press 2ndF and ZOOM to get the FORMAT menu, tap E STYLE1 and then double tap 2 *Dot*, as shown below. This will change the graphing mode from connected points to individual dots.

Press GRAPH to obtain a fresh image of randomness like this one:



Of course, your image will not be exactly the same as this one, since your calculator will (almost certainly) generate different random numbers than those generated above.

Activity 6

1. Graph the *random* function as shown above with the style set to *Connect*. (Press 2ndF ZOOM to check the style in the FORMAT menu.) Compare your graph with your partner's and with the images above. Notice especially the shape of the graph near the vertical axis.

   Now zoom out on your graph (using ZOOM 4). Describe what happens to the graph. Compare the results with your partner. Why does the graph change in this way?

   Now zoom in on the new graph (using ZOOM 3). Describe what happens to your graph. Why does the graph change in this way?

   Usually when you zoom out and then zoom in on a graph, the final result is the same as the original. Notice that in this case the graph of the *random* function is *not* the same as the original graph. (Check near the vertical axis to see the differences.) Why is it different?

   Repeat the zoom in and zoom out steps a few times until you are confident that you understand what is happening.

2. Graph the *random* function and then zoom out four times in succession. Explain what happens.

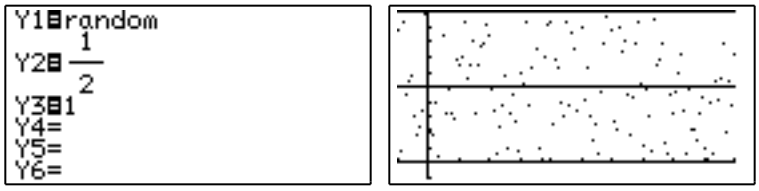
   Then zoom in four times in succession to restore the original window.

3. Change the style of the graph from *Connect* to *Dot.*

   Repeat Activities 1 and 2 with the new style.

4. Some people prefer to see the *random* function plotted as individual dots, as it is easier to see the sizes of the numbers generated. Make sure your calculator style is set to Dot.

   Graph the *random* function and also graph horizontal lines at a half and one to divide the screen in two as shown below.



   Check for yourself that about half the numbers fall in each half of the screen. (There are probably too many to count them easily.) Compare with your partner.

   Zoom out and check again.

   Zoom in and check again.

5. Repeat activity 4, but graph some extra functions so that you can check four quarters of the screen rather than just two halves.

   Count how many points are plotted in each quarter.

   Compare your results with your partner.

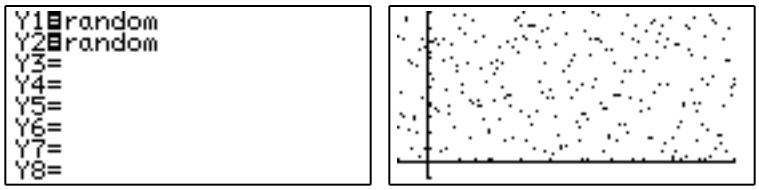
6. Repeat activity 5, but first change the style of the graph (STYLE2) from *Sequence* to *Simultaneous,* as shown in the FORMAT menu below.

   This change will allow you to see how the random numbers spread over the four quarters at the same time they are generated by the calculator. You should find from this dynamic display that the results are fairly evenly spread.

   When you have finished, change the style back to *Sequence* for later activities.

7. Make sure the graph styles are set to *Sequence* and *Dot* for this activity.

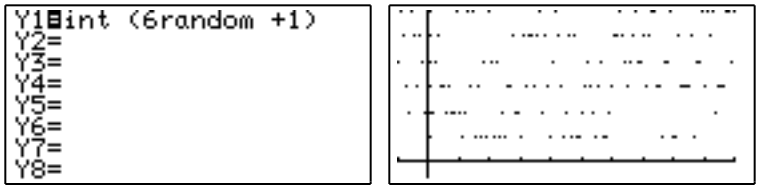
   When you graph the *random* function, the points plotted are rather sparse, since there is only one value plotted for each of the (127) positions across the screen. You can plot further points by defining a second random function as shown on the screens below.



   The calculator will generate a fresh set of random numbers. Watch the screen carefully to see what is happening. Notice that there are about twice as many points as originally.

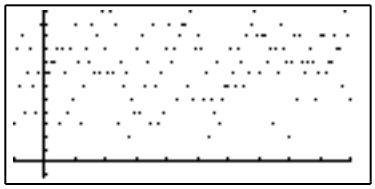
   You may like to define even more random functions (up to ten are possible) to see how evenly distributed the random numbers are.

8. Other images of randomness can be constructed as well. You are limited only by your own imagination. For example, the display below shows a graph of the standard dice function in *Dot* mode, with a suitable vertical scale.



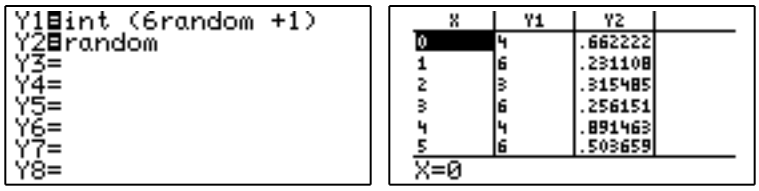
The graph shows that there are about the same number of each of the six values occurring.

In contrast, the graph below shows a simulation of rolling a *pair* of dice, using the function *int (6random +1) + int(6random +1)* and with a suitable vertical scale. Notice that, this time, each of the possible results from {2,3,4, ... ,12} does not occur as often as each of the others. Try this for yourself.



Examine some more images of randomness for yourself.

As well as graphical displays, the calculator can show numerical displays of random information. The easiest way to see how this works is to construct a table of values for a function, using the TABLE key instead of the GRAPH key. For example, below is a table of some values of the standard dice function and the *random* function together in a table.



Activity 7

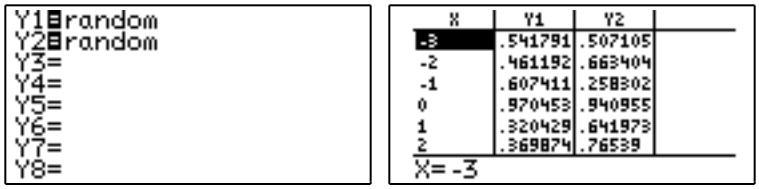
1. Make a table of the *random* function. Note carefully the top three numbers in the table. (Write them down.)

   Now move the table cursor up one space using the up arrow.

   Check the numbers in the table. Compare them with the previous ones.

   What has happened? Why do you think that would happen?

2. Make a table of values for a *pair* of random functions, as shown below.

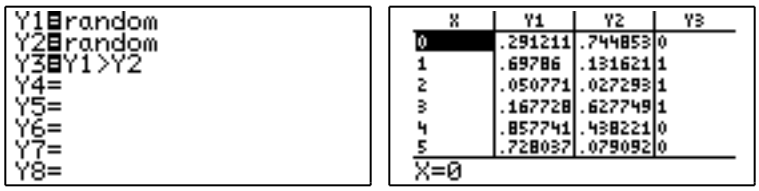


There are six pairs of values in the table. Notice that in three of the six cases shown above, the first values is larger than the second. Is this also the case for your table? Compare with your partner.

Move the cursor up one place and count again.

Repeat this experiment a few times. Describe your results.

3. You might expect it to be easier to get the calculator to do the comparison for you in Activity 2. The screen below at left shows one way of doing this. (The Y1 and Y2 variables are available by pressing the VARS and ENTER keys; the inequality is obtainable from pressing MATH and then tapping F.)



The variable Y1>Y2 is a logical variable. It has a value of 1 when Y1 is greater than Y2 and a value of zero when the opposite is true (i.e. when Y1 < Y2).

Check the table above carefully. Then display one for yourself in the same way.

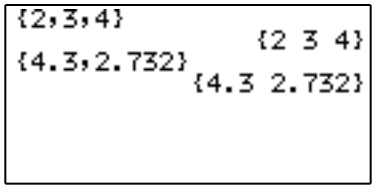
Can you explain why this method *doesn't* work?

In later parts of this unit, you will see other ways to display random information and to analyse it more carefully than we have done here.

8.4 Generating random data sets

To study random phenomena, it is necessary to look at many data points and not just a few. The main reason for this is that the patterns evident in random data are not clear without a lot of data. Things only become clear in the long run. Your calculator can be used to generate sets of random data and also to analyse them to see any patterns involved.

On the EL-9600 calculator, sets of data are organised into lists. A list is an ordered set of objects, in this case numbers. Examples of lists are {2,3,5} and {4.3,2.7234}. On the calculator, lists can be entered using the curly brackets (which are just above the parentheses on the keyboard), with the elements separated by commas. The screen on the next page shows the two lists described here.



Notice that the calculator shows lists without commas, even though you must enter them using commas to separate elements.

To generate pairs of dice tosses, you can construct a list of two elements, using the familiar function, *int (6random +1)*. The screen shows this (even though not all the list definition can be seen because of the width of the screen). Each time ENTER is pressed, another pair of dice tosses is simulated.

The complete list definition in this case was {*int (6random +1),int (6random +1)*}.

Activity 8

1. Use the command for generating a pair of dice tosses a total of 20 times. Count how often a double is simulated – i.e. the two dice show the same number.

   Compare your results with someone else's.

   Repeat the experiment and see if the results are similar the next time.

2. Use the command for generating a pair of dice tosses a total of 20 times. Count how often both of the two dice show 1, 2 or 3.

   Compare your results with someone else's.

   Repeat the experiment and see if the results are similar the next time.

3. Write a command to simulate tossing a pair of coins, with a one representing a head and a zero representing a tail.

   Simulate a number of tosses (at least 20) and check how often a pair of heads is thrown.

   Compare your results with those of other people.

4. It has been suggested that, when three standard dice are thrown, about half the time all three dice will show different numbers. Write a command to simulate tossing three standard dice. Use it a number of times to see whether or not this suggestion seems reasonable.
   Compare your results with your partner's.

5. The weather forecaster on television said that there was a 70% chance of rain on each of the next three days. Use a random command to simulate the weather for each of the next three days. Use your command to estimate how likely it is that there will be nor rain at all on the next three days.

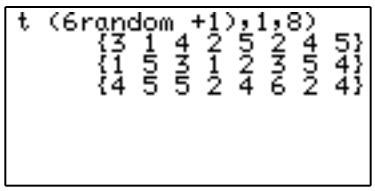
   Compare your results with your partner's.

It would be rather time-consuming to generate a large list in a similar way to those shown above. Fortunately, there is an easier way to generate a list when each of the elements of the list can be described by a formula. This way uses the *sequence* command for lists, shown below, after first pressing the LIST key (i.e. the 2ndF key and then the multiplication key).

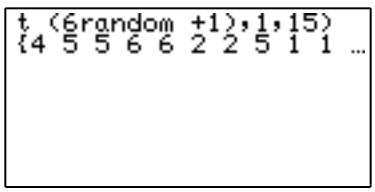
The sequence command (abbreviated on the calculator to *seq*) requires you to specify three things: the function concerned, a starting value and a finishing value. So, to generate a sequence of eight dice tosses, the following command is needed:

seq(*int* (6*random* +1),1,8)

In this case, the starting value is 1 and the finishing value is 8. When this command is entered into the calculator, a set of eight dice tosses is simulated each time ENTER is pressed. The screen below shows three sets of eight dice tosses. In the first two sets, there was no six tossed, while the third set does not have either a one or a three. (Again, notice that the screen shown here is not wide enough for us to see all of the *sequence* command.)

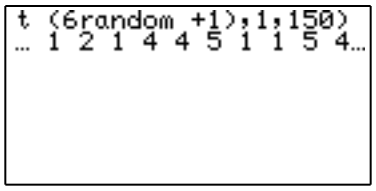


To change the number of elements generated in a set, it is necessary only to change the final number in the command. Since it takes a while to enter a complicated command like this, it is often easier to recall the previous command and to edit it than to start all over again. To do this, first press 2nd ENTRY (i.e. the yellow 2ndF key followed by (–), the opposite key.) Then the command can be edited using the left and right cursor keys and the DELete key.

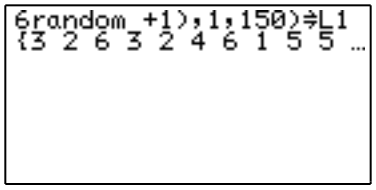


Notice that if you generate a data set wider than the screen can show, only the first few elements will be showing. In the screen above, only the first ten of the fifteen dice rolls are shown, followed by three dots ... , which is a signal that some data are not showing on the screen. To see the rest of the data set, use the cursor arrow to go to the right. Later you can use the left arrow to return to the start.

It is inconvenient to have to use the cursor arrows to see all of the data generated in a large set and sometimes you will want to store a set of data for analysis. Consider, for example, the data set below, consisting of 150 dice rolls. As you can tell from the three dots at each end of the list, the data extends in each direction. It would take a long time to see all of this data! So we need an easier way.



Fortunately, you can save a list into one of the calculator's six data lists, labelled L1, L2, L3, L4, L5 and L6. The command is the same as that used for storing a value into a variable memory – the STO command, followed by the name of the list. In this case, the set of 150 dice rolls can be stored into L1 automatically by adding the command STO L1 after the *sequence* command, as shown below. (To get L1, press 2ndF followed by the 1 key.)



Once the data are stored in the list, they can be recalled to the screen by entering the name of the list, as shown in the first screen below. They can also be analysed in various ways, using the LIST MATH operations, as shown in the middle screen below.

For example, the *mean* command finds the mean value of all the elements in the list. In this case the mean of the 150 simulated dice tosses stored in L1 is about 3.51, as the third screen above shows.

Activity 9

1. Use the *sequence* command shown above to generate 100 dice tosses and store them into list L1.

   Then use a *mean* command to find the mean score of the 100 simulated dice tosses. The result is quite likely to be close to 3.5, as in the example above.

   Repeat this experiment a few times, and see how consistent your results are.

2. Use a sequence command to generate 100 coin tosses, representing heads with a one and tails with a zero. Store the set of coin tosses into list L1.

   Use the *sum* command, *sum(L1)*, to count how many heads were simulated in the 100 coin tosses.

   Do this several times and study the results carefully. Compare your observations with those of your partner.

3. Generate at random a set of six numbers for *Lotto* (i.e., whole numbers between 1 and 45). Compare your set of six numbers with the winning numbers published in a recent national newspaper.

   You will notice that sometimes the numbers generated will not all be different, as required for *Lotto*, since the calculator generates each number separately and is not restricted to generating a fresh number each time.

4. Write down what you might expect from a random set of ten tosses of a fair coin. (Don't use the calculator.) Compare your 'random' set with your partner's.

   Now count how many runs of three or more heads in a row you have. Compare this also with your partner's.

5. Study carefully these commands for simulating coin tosses and try them on your calculator:
   int (*random* + 0.5)
   seq(*int* (*random* + 0.5),1,10)
   sum(*seq*(*int* (*random* + 0.5),1,10))
   seq(*sum*(*seq*(*int* (*random* + 0.5),1,10)),1,8)

   Explain what each command does.

6. Use the calculator to simulate some random sets of 10 tosses of a fair coin . For each set, note whether or not it has a string of three or more heads (i.e. ones) in a row. How often does this happen with your data?

   Compare your result with your 'random' sets in question 4.

   Generate some more sets of ten coin tosses. This time check how many have either a run of three or more heads or have a run of three or more tails (or both). Compare your observations with your partner.

   How successful were you at writing down a 'typical' random set in question 4?

7. A certain professional basketball player seems to get about 85% of her free throw shots in the basket.

   What command will simulate a free throw (with 1 representing a basket and 0 a miss)?

   Simulate a set of five throws to represent a recent game in which she had five free throws.

   Do this several times and try to summarise your results. For example, how often does she successfully complete all five baskets? Compare your summaries with other people's.

8. Some children's games require you to throw a six before you start. Use your calculator to simulate some dice rolls to see how long it takes to get a six. You will need to repeat this several times to get a good idea of the typical results.

9. Imagine two evenly matched tennis players, such as Monica Seles and Martina Hingis. (Or yourself and your friend.) Use the calculator to simulate a win by Monica (1) or Martina (0) in a game, with each one winning about half of the games.

   Study how often someone wins a streak of games (like three or four or five in a row). Work with a partner.

   Sports commentators sometimes claim that one player is superior to the other, because she has won the last three games they have played. Do you think this is a sound opinion?

In the next section, you will see other ways in which the data can be analysed more systematically, using some of the calculator's built-in features.

8.5 Analysing random data sets

When a set of data has been generated at random, it is important to study it carefully to understand its significance. The EL-9600 calculator allows a data set to be analysed in a number of ways, both numerically and graphically.

To illustrate some of the possibilities, we will simulate tossing a pair of standard dice, using the command

seq(*int* (*6random* +1)+*int* (*6random* +1),1,100) STO L1

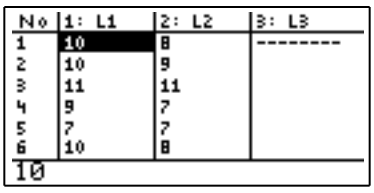
Study this command carefully to see how it works before you enter it into the calculator.

After the calculator has finished generating the data, press 2ndF ENTRY and edit the command to generate another 100 pairs of dice tosses, this time stored in List L2:

seq(*int* (*6random* +1)+*int* (*6random* +1),1,100) STO L2

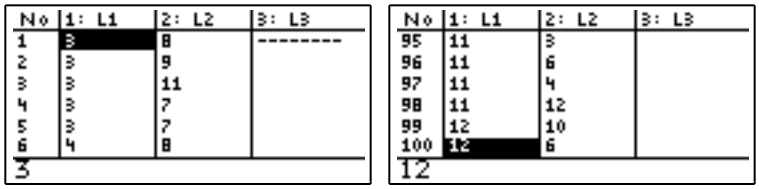
Each of these commands will take about 10 seconds to complete, since the calculator has to generate 200 random numbers for each.

The results can be seen in the two lists by pressing STAT and then tapping A twice. Your results will of course not be the same as those shown below:



It is possible to scroll each of the two lists to get a good idea of the sums of the pairs of dice, but this is rather tedious. A better way is to sort the data. To do this return to the home screen and press STAT followed by tapping B and then double tapping 1 to sort the data into ascending order. You need to complete the command by telling the calculator which list to sort and then pressing ENTER. The screen below shows how to sort L1:

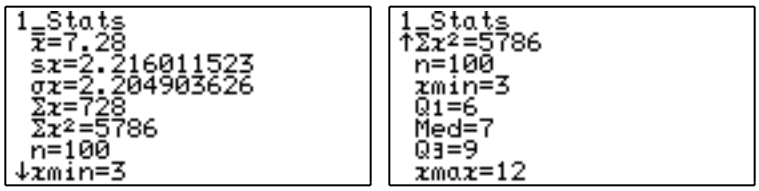
If you now look at the data again using STAT, you will see that the 100 scores in L1 have been sorted from lowest to highest. The two screens below show the top and the bottom of the sorted list. (A quick way to move from top to bottom and vice versa is to press 2ndF and then the down or up arrow key.)



In this particular case, although it is possible to get a score of 2, the lowest score was 3. It is also possible to get a 12, which happened on two of the simulated throws for L1.

Although you can get a numerical analysis of the data using the LIST operations described in the previous section, a more efficient way is to use the STAT CALC menu as shown below in the home screen.

The one-variable statistics (abbreviated on the calculator to *1_Stats*) for L1 take a few seconds to compute after ENTER is pressed, and occupy more than one screen, as shown on the screens below. (Notice the arrow in the bottom left of the first screen, showing that further information is available by pressing the down arrow):



As well as the minimum and maximum scores (of 3 and 12 respectively), these screens show the mean, median and standard deviation of the 100 simulated dice scores. In addition, both lower and upper quartiles (6 and 9 respectively) and some summary data are shown. Together, these present

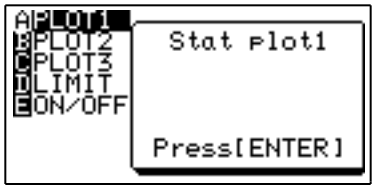
Activity 10

1. Use the screen immediately above to check that the mean of the dice throws has been calculated correctly.

2. Repeat the operations above on the second set of data in your calculator, stored in L2. Write down the main similarities and the main differences between the two data sets. Discuss your observations with your partner.

As well as a numerical representation of data, a graphical representation will help you to understand what has been found. In this case, the two most appropriate graphs are box and whisker plots (usually called box plots) and histograms. First we will explore the box plots. Before you start, press *Y*= and delete any functions stored in the calculator, using the CL key.

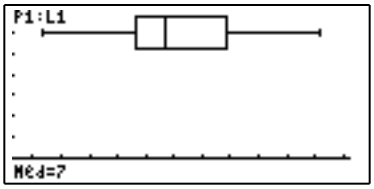
To draw a box plot for the data in L1, first press STAT PLOT (i.e. *2ndF* and then the *Y*= key) and press ENTER to select one of the three plots. In the screens below, PLOT1 has been selected.



Use the cursor to turn the plot on (in the first line), to select X (for univariate, or single-variable data) in line 2, to enter L1 into line 3 and to DELete anything from the Freq(uency) space in line 4. In each case, press ENTER after making your choice. Your screen should now look like that on the left below (but may show a different graph at the bottom).

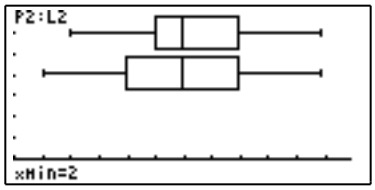
To select the kind of graph you want (in this case, a box plot), move the cursor down to the word GRAPH and then press 2ndF and Y= again to see the menu shown in the middle screen above. Tap E and double tap 1 to select the Box graph. The result should now look like the third screen above.

To draw the box plot, press ZOOM 9, which will automatically adjust the screen window to suit the data. The result will be a box plot of the L1 data such as the one shown below.



If you press the TRACE key and move the left and right cursors, you can see that the box plot shows the five important data points (the minimum and maximum scores, the median and each of the lower and upper quartiles). The box plot here shows that the data are skewed a little towards lower values, since the median (7) is closer to the lower than to the upper quartile and is not in the centre of the data.

You can compare the two sets of simulated data (in L1 and L2) by defining a second STAT PLOT for L2 and then pressing ZOOM 9 again. Both of the box plots will be drawn together, so that it is easy to make comparisons. The graphs below show that the distribution of scores was a bit more symmetrical for the second list, and even included scores as low as 2.



(To choose which graph to trace, use the up and down arrows after pressing TRACE.)

Activity 11

1. Draw a pair of box plots together, as described above, for your own data in L1 and L2. use the box plots to compare the two data sets. What extra information is provided by the graphs that you did *not* write about in Question 2 of Activity 10?

2. Draw a box plot of the data in L1 as shown above using ZOOM 9.

   Then turn PLOT1 off and draw a box plot of L2 by itself using ZOOM 9.

   Explain why it is difficult to compare the box plots drawn in this way.

3. Experiment with drawing box plots for L1 and L2 by first adjusting the WINDOW variables manually (i.e. rather than using ZOOM 9). What happens if the *x*-values chosen are not wide enough?

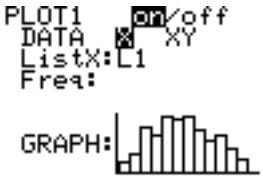
4. Generate a third set of 100 simulated throws of a pair of standard dice and store it in L3. Compare your three sets of data using box plots on the same screen.

5. Generate a set of throws of a pair of 10-sided dice. Graph the results, using a box plot. In what ways are they different from the results for 6-sided dice? In what ways are they the same?

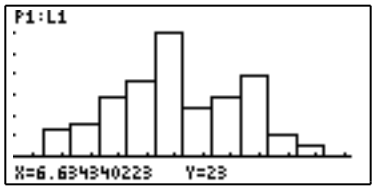
6. Experiment with other ways of simulating throwing dice. For example, what would you expect the sums of scores of three 6-sided dice to give? Compare your prediction with your partner's.

   Then each generate and graph some data to test your predictions.

A second way of analysing univariate data is to draw a histogram. This shows each of the scores obtained and the frequency with which they occurred. Once you know how to draw a box plot, drawing a histogram should not prove difficult: the only difference is in the final step of choosing a graph type, as shown below for the L1 data. Before you start, turn off any STAT PLOTs you are not using.

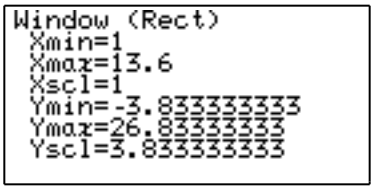


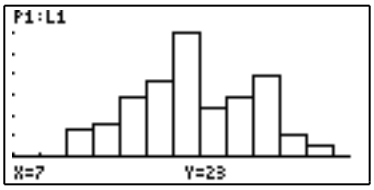
After pressing ZOOM 9 (for automatic scaling of the *x*-axis), the resulting histogram shows the complete distribution of simulated dice throws, as shown below. As for box plots, the histogram can be traced to see the frequency for each category.



Notice that the intervals chosen for the horizontal axis seem to be non-integers, and the columns do not quite match the *x*-axis, even though the data are in the set {2,3,4, ... , 11,12}. This occurs because of the way the calculator chooses scales when the ZOOM STAT option is chosen.

A (good) alternative to using ZOOM STAT is to choose *x*-values for yourself that will suit the data and the screen. Since there are 126 pixel spaces across the screen, a range of 12.6 (a number which divides evenly into 126) is especially convenient. The result is shown below, after pressing GRAPH instead of ZOOM STAT.





Activity 12

1. Draw a separate histogram for each of your data sets from L1 and L2. How do the two sets compare?

   Which scores are most common when a pair of dice are thrown? Which scores are least common?

Notice what happens if you draw the two histograms together (by defining each of STAT PLOT1 and STAT PLOT2 and then using ZOOM 9). While it is helpful to draw a pair of box plots, it is not helpful to draw a pair of histograms.

2. Trace a histogram carefully, using the second WINDOW shown above (i.e. with *x*-values from 1 to 13.6). What does the *x*-value tell you each time you trace? What does the *y*-value show? How are these different from conventional histograms?

3. What do you think will happen if you try to draw a histogram with STAT PLOT 2 and a box plot with STAT PLOT 1? Test out your prediction using the data in the calculator.

4. Consider again the basketballer described in question 7 of Activity 9. use the calculator to simulate 50 sets of 5 free throws and count how many were successful. A command to do this is:

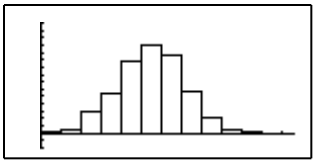
   seq(*sum* (*seq* (*int* (*random* + 0.85),1,5),1,50) STO L1

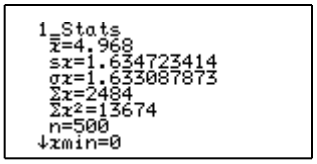
   Graph the resulting data in a histogram.

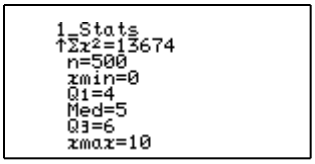
   Describe your results carefully. Compare them with your partner's.

5. Repeat question 5 for basketballers with different free throw success rates. Try one with a higher rate and one with a lower rate. Investigate what happens.

6. The data graphed below show a simulation (500 separate cases) of the results of tossing 10 fair coins and counting how many heads were observed.







   What command could be used to generate such data?

   Do this for yourself (but be patient! It will take a long time to generate, summarise and graph the data.) How do your results differ from those shown here? How are they the same?

7. A student attempting a multiple-choice test decides to guess the answers instead of trying to work them out. If there are four answers to each question, what command will simulate the number of correct answers to 20 questions?

   Use this command to generate a set of 50 students guessing in this way. Draw a histogram of the results. How many students scored 10 out of 20 correct or even better?

   Use your histogram to write some advice for other students about the advisability of guessing answers in this way.

8.6 Random sampling

An important use of randomisation is for random sampling. In many practical situations, it is not

taken. The process of making inferences from samples to populations is the single most important aspect of statistics, in fact.

This is the basic idea underlying surveys of opinion, for example. Provided a good enough sample of people's opinions is taken, a good estimate of the population's opinion can be obtained. The crucial property of samples chosen to represent a population is that they are taken at random – i.e. that each possible sample has the same chance of being chosen. It is much easier to ask a random sample of about 2000 people their opinion regarding the next election, than it is to ask the entire electorate. It is also much less expensive to get the information and it is possible to get a result much more quickly.

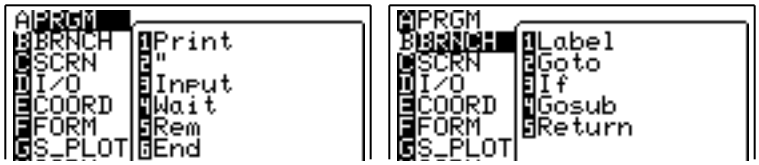
You can use your calculator to study some of the properties of random sampling. To do this, it is necessary to use a small program to select random samples from a population of data. The program called SAMPLE given below will allow you to do this.

SAMPLE
Print "DATA MUST BE IN L1
Input N
N → dim(L2)
dim(L1) → M
1 → J
Label LOOP
int (Mrandom +1) → K
L1(K) → L2(J)
J+1 → J
If J≤N Goto LOOP
Print mean(L2)

The easiest way to put this program into your calculator is to use the LINK options to electronically transfer the program from a computer or from another calculator. To do this, you must have the appropriate cables and computer software (the Sharp CE-LK1 accessory pack).

Alternatively, you can enter the program into the calculator's program area directly. First press PRGM (i.e. 2ndF MATRIX) and select NEW to start. Enter the title, followed by ENTER and then the program steps shown above.

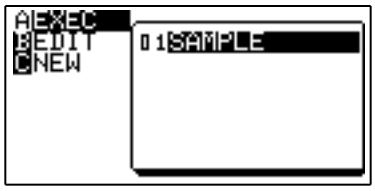
Press ENTER at the end of each line of the program. Some of the commands in the program are in the PRGM and BRANCH menus, shown below. These menus become available in programming mode after pressing PRGM (2ndF MATRIX).



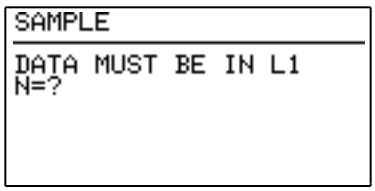
You will also need to enter the dimension command (abbreviated to *dim*), which is in the LIST OPE menu, as shown below. The dimension of a list is the number of elements in the list.

The program selects a random sample of a certain number of elements from a population of data, which must be stored in list L1 before you start. The program stores the random sample into list L2 and prints the mean of the random sample on the screen.

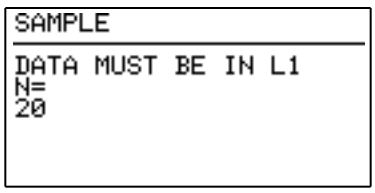
To illustrate the program working, 150 simulated dice throws were stored into L1. Then, using the PRGM menu, select EXECute and double tap the name of the program from the menu, as shown below.



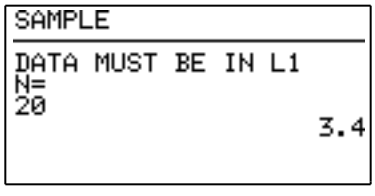
The program begins with a reminder that the population data must be stored in L1. It then asks for a value for N, the sample size:



To select a random sample of size 20, enter 20 and then press ENTER.

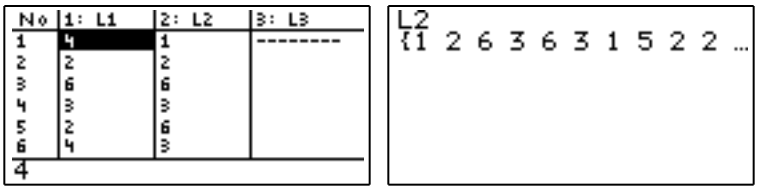


The calculator takes a few seconds to select the random sample, and then prints the mean of the sample on the screen. In the example shown below, the mean of the 20 sampled elements is 3.4.

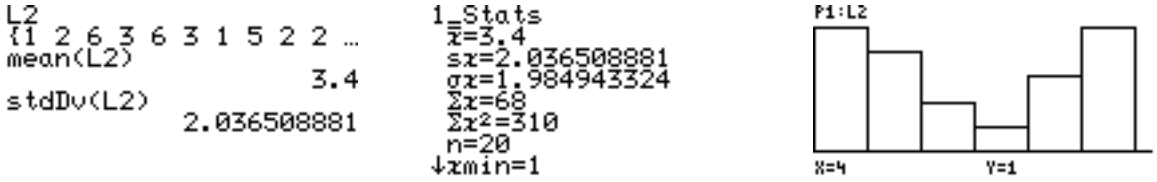


To select another (almost certainly different) sample, simply press ENTER again, and the calculator will run the program another time.

Each time the program is run, the calculator stores the random sample selected into list L2. Any data previously in list L2 is deleted first. You can see the details of the most recent sample chosen by the calculator by examining the contents of List 2, either in STAT mode or in the home screen in the usual ways, as shown below.



If you want to, you can analyse or display the sample in detail in STAT mode or in the home screen using the LIST operations, as the examples below show.



The histogram shows that the particular random sample chosen here has a surprising distribution, with only one value of 4 chosen. It is only in the long run that we expect random samples to be similar to the populations from which they have been chosen; it is not unusual for small samples like this to have some odd features such as this. (This explains in part why people prefer to take large samples when they can afford to do so.)

Activity 13

1. Obtain the heights (in centimetres) of all the students in your class and enter them into the calculator as L1. Find and write down the mean and standard deviation of the heights.

   Now use the calculator to select a random sample of five heights from the class. Write down the mean of the heights.

   Repeat the sampling process several times until you have a good sense of the range of sample means that occur. It will be efficient to combine your results with your partner's.

   How do the sample means compare with the population mean? (i.e., the mean height of the whole class.)

   Compare the standard deviations of the samples and the population.

2. Repeat activity 1, but with a larger sample. Try $n = 10$ or $n = 20$. Compare the results of taking larger samples with those of smaller samples.

3. You may have heard of the Birthday Problem: How likely is it that two people in a room will share the same birthday? (Not the year, just the day.) Of course, it is more likely if there

You can simulate this problem by putting all the birthdays into L1 and then using the SAMPLE program to simulate a certain sample size of people in a room.

To put the birthdays into L1, it is easiest to use a sequence command:

seq(X,1,365) STO L1

(Notice that this procedure doesn't include leap years.)

Then use the SAMPLE program to simulate a set of 20 people in a room. Check L2 to see if there are any pairs of 'people' in the list with the same birthday. (An easy way to do this is to first sort the data.)

Compare your results with some other people.

Now repeat the simulation for a set of 30 people in a room.

How many people in the room seems to be enough to ensure that most of the time there is at least one pair of people with the same birthday?

4. It is possible to edit the SAMPLE program to automatically graph each sample as it is produced. The edited program is shown below. The last line of the original program has been replaced with the last four lines below in the revised program GSAMPLE.

GSAMPLE
Print "DATA MUST BE IN L1
Input N
N → dim(L2)
dim(L1) → M
1 → J
Label LOOP
int (Mrandom +1) → K
L1(K) → L2(J)
J+1 → J
If J≤N Goto LOOP
Plt1(Hist,L2)
Zm_Stat
DispG
Wait

To use this revised program, first turn off any other STAT PLOTs. When the program is run, the sample will be taken and then graphed. The histogram of the sample will remain on the screen until a key is pressed.

8.7 In the long run

One of the most important aspects of random phenomena is their unpredictability. But, paradoxically, another one of the most important aspects of random phenomena is their predictability! In this section, we consider the simple example of throwing a standard die to

When a coin is tossed once (or this is simulated with a random number generator on your calculator), you cannot successfully predict the result. But when a coin is tossed many times, you can predict the results much more confidently. In the short run (i.e. in just one toss), the result is unpredictable; in the long run (i.e. after many tosses), the result is predictable. We will demonstrate this with a short program.

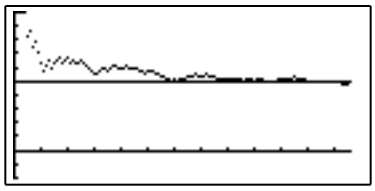
The program LONG RUN is shown below. As previously, it is easiest to transfer the program into your calculator using a cable attached to another calculator or a computer on which the program has already been placed. If these are not available, you can enter the program directly into your calculator's program area, by first pressing PRGM (i.e. 2ndF MATRIX).

LONG RUN

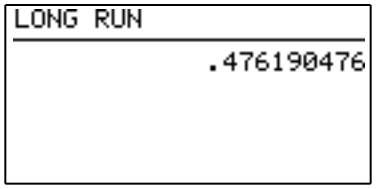
```
ClrDraw:H_line 0.5
(Xmax - Xmin)/126 → D
0 → S:0 → K:Xmin → X
Label TOSS
int (random +0.5) → T
If T ≠ 1 Goto NEXT
S+1 → S
Label NEXT
K+1 → K
PntON(X,S/K)
X+D → X
If X<Xmax Goto TOSS
Wait
Print S/K
```

This program simulates tossing a fair coin 127 times and evaluates the proportion of the tosses that are heads after each toss. The result is plotted on a graph, so that you can see what happens in the long run. The program uses three variables. The number of successful tosses is labelled *S*, the number of tosses so far is labelled *K* and the result of each toss is labelled *T* (with 1 for a head and 0 for a tail).

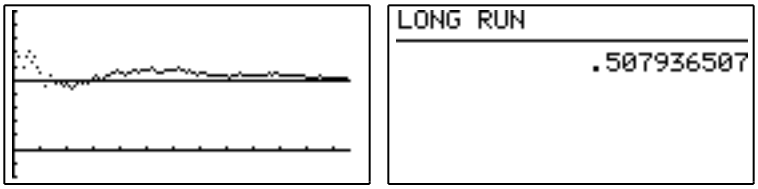
When you run LONG RUN (using EXEC in the PRGM menu), the results of the simulation are shown on a screen like the one below. Each dot plotted shows the proportion of the tosses so far that have been heads.



Notice that in the short run (the first few tosses), the proportions change fairly quickly and are not very close to a half. But in the long run (after many tosses, towards the right of the screen), the proportion is quite close to a half, which is shown by the horizontal line drawn at the 50% mark. After the program has finished plotting, press ENTER to see the proportion of the 127 tosses that are heads. In this case, it was about 0.476 or 47.6%, as shown over page.



To use the program again, you just need to press ENTER again. The screen below shows another run of the program, with similar results, but not exactly the same.



Activity 14

1. Run program LONG RUN a few times. Record the results each time so that you can compare them with your partner.

2. Run program LONG RUN. Look carefully at the dots and imagine that each dot is joined to the next dot with a line. How many times does this line cross the horizontal 50% line? (In the screens above, the number of crossing times is 1 and 4 respectively.)

   Record the result a few times and compare with others.

3. You can use program LONG RUN to simulate other random events. For example, suppose you have a biased coin that shows heads 60% of the time. You can simulate this by changing one line of the program. Change the 5th line from

   int (random +0.5) → T

   to

   int (random +0.6) → T

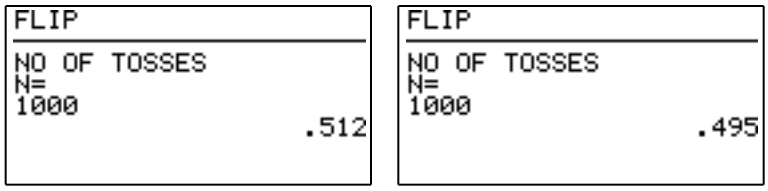
   by using the EDIT command in the PRGM menu. (You might also want to change the first line ClrDraw:H_line 0.5 that draws a horizontal line by changing 0.5 into 0.6, but it is not necessary to do this.)

   Examine some other events in the long run in this way. For example, investigate how many sixes appear when a dice roll is simulated, by changing the 0.5 in the 5th line to 1/6.

You may have wondered why the program LONG RUN uses 127 simulated events. The reason is that the screen on the calculator has only 127 pixels from left to right and so only 127 results can be plotted. However, to see what happens 'in the long run' it is probably a good idea to look at many more cases than this. The short program FLIP below can be used to investigate the results for *any* number of successive coin flips.

FLIP
Print "NO OF TOSSES
Input N
0 → S:1 → K
Label TOSS
int (random +0.5)+S → S
K+1 → K
If K<N Goto TOSS
Print S/N

When you run the program, you will need to say how many coin tosses to simulate. The screens below shows what happened on two separate runs of 1000 tosses.



The proportion of 0.512 printed on the first screen shows that 512 of the 1000 tosses resulted in heads. When the program was run a second time, as shown on the second screen, 495 heads were simulated, to give a proportion of 495/1000 = 0.495 or 49.5%. In each case, although the model for the coin is that 50% of the time it will produce a head, neither of these long run simulations produces exactly 50% heads. However, each is fairly close to 50%, as we would expect.

Be careful with simulations like these. The calculator is quite fast, simulating about ten tosses per second, depending on the state of its batteries. But that means that 1000 tosses takes about one and a half minutes. If you put in a *very* large number of tosses, such as a million, you will have to wait a *very* long time to finish. Very long runs will also flatten your calculator's batteries more quickly than usual, too.

If you want to stop a program when it is running, press the ON key.

Activity 15

1. Use program FLIP. Compare your results with those of other people.

2. How long would a million coin tosses take to simulate, using this program? (*Don't* test your answer by trying it!)

3. You can use program FLIP to simulate other events, by adjusting the 5th line of the program that generates a random number and adds the resulting successful tosses:

   int (random +0.5)+S → S

   For example, if you change the command to the following line

   int (random +1/6)+S → S

   you can simulate rolls of a die to get a six, which should happen about one sixth of the time for

8.8 A Monte Carlo procedure

One of the uses of randomisation is to solve problems that are too hard to solve otherwise. Solutions that rely on random simulations are sometimes called Monte Carlo solutions, named after the famous European gambling casino. Monte Carlo solutions to problems have become popular in the last 50 years, since technologies like computers and calculators have been invented to generate random processes quickly and efficiently.

Here is a problem that requires rather sophisticated mathematics to solve, but for which a good solution can be found by using a random procedure:

> *A breakfast cereal maker decides to include a card showing a pop star in each packet of breakfast cereal sold. There are six different cards. Anyone who collects all six cards will win a prize. How many packets will people probably need to buy in order to win a prize?*

Before reading any further, write down what you think the answer to this question is. Show your answer to your partner and tell each other how you made your guess.

One way to solve this problem is to simulate the processes involved and to study the simulations. First, it is necessary to make the assumption that the card in each breakfast cereal pack is equally likely to be any of the six cards. This would only be the case if the manufacturer made the same number of each card and then distributed them randomly to packets.

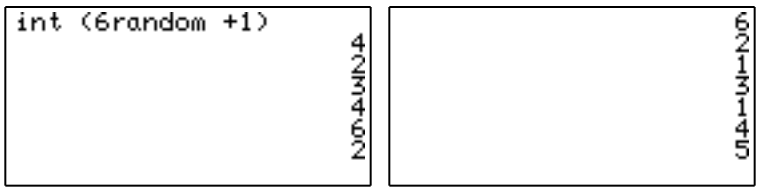
With this assumption, you can simulate the process of buying a packet of breakfast cereal and looking at the card by tossing a standard die. The six faces of the die can be used to refer to the six pop stars. As soon as you have a complete set of the six numbers (i.e. the six pop stars), you will win a prize.

A second assumption is needed here: that the cards are not swapped with other people to make up a set.

You can simulate all of this on the calculator by generating random dice throws and counting them until you have a set of all six numbers. The random dice throw command is

$$int\ (6random\ +1)$$

The screens below show one simulation. Notice that the second screen is a continuation of the first one, with the 6 and 2 cards overlapping.



Altogether, it took only eleven cereal packets (presses of the ENTER key) until the final number (pop star 5) appeared.

To solve the problem (at least approximately), you will need to repeat this process many times and find some sort of average or typical number of packets needed. One good way of doing this is for

Activity 16

1. What is the least number of packets possible to get a complete set of breakfast cereal cards?

   What is the most number of packets that could be needed?

2. Use the calculator to repeat the above simulation five times. Record the number of cereal packets needed each time. Compare this with your partner. Find the average number of packets needed.

   Compare your results with the guess that you made at the start of this section. Revise your guess if necessary.

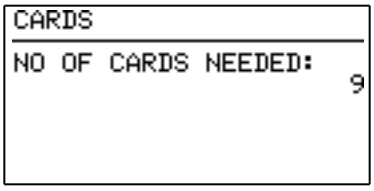
3. How many simulations do you think you will need to be confident of your result? (For example, confident enough to say something like, "Only about one time in ten would you need to buy more than ... packets.")

Another way to use the calculator to simulate this problem is to use a short program. The advantage of using a suitable program is that the calculator will quickly and efficiently do the necessary counting. One possibility is the program CARDS shown below.

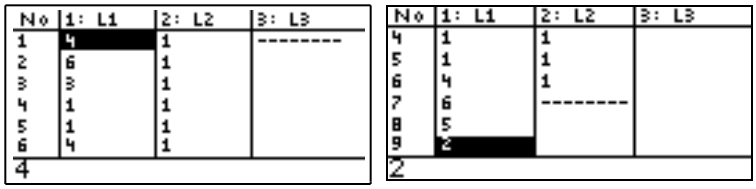
CARDS
1 → dim(L1):6 → dim(L2)
0 → N
fill(0,L2)
Label NEXT
N+1 → N
int (6random +1) → X
1 → L2(X)
X → L1(N)
If sum(L2)<6 Goto NEXT
Print "NO OF CARDS NEEDED:
Print N

This program stores the random numbers that are generated into list L1 and uses list L2 to keep a check on which cards have been chosen so far: a one or a zero in each of the six elements of list L2 indicates whether or not the corresponding number has been chosen. When all six cards have been chosen, the sum of list L2 is six and the program stops and prints out the result, showing how many cards were needed.

The screen below shows what happened when program CARDS was used once.



In this case, after the program had finished, the lists L1 and L2 could be seen by pressing STAT followed by A and ENTER. The screens below show that the ninth card chosen was a 2, which completes the set, since then each of the six elements of list L2 have the value of 1.



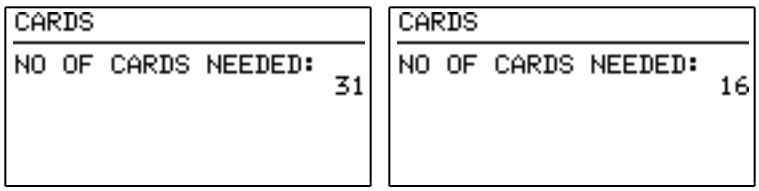
No	1: L1	2: L2	3: L3
1	4	1	--------
2	6	1	
3	3	1	
4	1	1	
5	1	1	
6	4	1	

4

No	1: L1	2: L2	3: L3
4	1	1	
5	1	1	
6	4	1	
7	6	--------	
8	5		
9	2		

2

The next two attempts at running the program are shown below.



CARDS

NO OF CARDS NEEDED: 31

CARDS

NO OF CARDS NEEDED: 16

It is much quicker and more reliable to use the program than to conduct the simulation manually.

Activity 17

1. Use program CARDS several times (at least twenty times), recording the number of cards needed each time. Find a suitable average of your results.

   How do your results compare with those obtained earlier (in Activity 16)?

2. A major advantage of a program is that it is fairly easy to change. For example, program CARDS can be adapted to suit the similar problem of finding the number of cereal packets needed to obtain a complete set of *eight* cards (instead of six).

   To do this, you need to EDIT the program to change all the sixes in the program into eights. Only the following three lines are involved:

1 → dim(L1):6 → dim(L2)	changes to	1 → dim(L1):8 → dim(L2)
int (6random +1) → X	changes to	int (8random +1) → X
If sum(L2)<6 Goto NEXT	changes to	If sum(L2)<8 Goto NEXT

   Make these changes to the 1st, 6th and 9th lines of the program and then use the revised program to explore the problem with eight cards.

3. Investigate the card problem if there are 12 cards to collect.

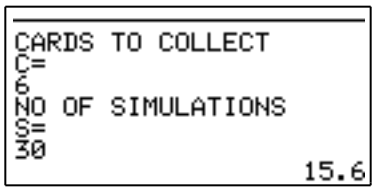
More sophisticated (and hence longer) programs can be used to solve problems of this kind. For example, program REPCARDS below allows you to specify how many cards are to be collected and how many separate simulations you want to conduct. The program repeats the simulation of CARDS several times over. At the conclusion of all the simulations, the mean number of cards required is printed and a histogram of the results is also drawn.

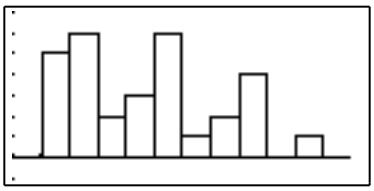
If you have collected all cards using REPCARDS you might see that it is similar to using CARDS

and the data are analysed at the end. In addition, the number of cards to be collected is a variable (*C*) rather than a fixed number.

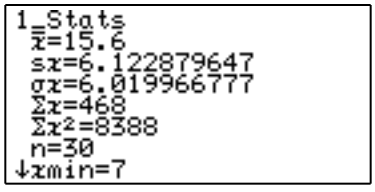
```
REPCARDS
Print "CARDS TO COLLECT
Input C
Print "NO OF SIMULATIONS
Input S
S → dim(L3)
1 → M
Label MORE
M+1 → M
1 → dim(L1):C → dim(L2)
0 → N
fill(0,L2)
Label NEXT
N+1 → N
int (Crandom +1) → X
1 → L2(X)
X → L1(N)
If sum(L2)<C Goto NEXT
N → L3(M)
If M<S Goto MORE
Print mean(L3)
Wait
Plt1(Hist,L3)
Zm_Stat
DispG
Wait
```

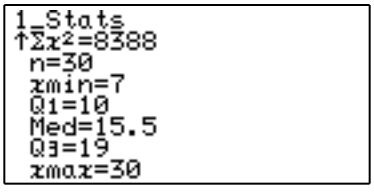
Here is a sample of the results of using program REPCARDS on the original problem of collecting six cards. In this case, 30 separate simulations were conducted:





After the results have been displayed, the *1_stats* command was used to study the distribution of results. The screens below show this.





Half of the results lie between the lower and upper quartiles (i.e. between 10 and 19), with the median and mean each close to 15-16 cards. Using the program allows results to be obtained

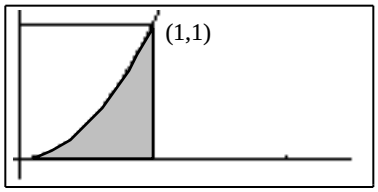
Activity 18

1. Use program REPCARDS yourself to see how different (and how similar) from the example shown above are your results for the case of collecting six cards.

2. Use program REPCARDS to investigate the likely number of cereal boxes needed to collect eight cards.

3. The manufacturer wants the cards to take people about 26 weeks to collect, using about one packet of cereal per week. Use the calculator programs to help you decide how many cards should be included in a set for this purpose. Work with a partner on this problem.

Finally, Monte Carlo solutions can be used for problems that are not related to random selection. An example is approximating a value for the area between the parabola defined by $y = x^2$ and the positive x-axis. The screen below shows a graph of the parabola between $x = 0$ and $x = 1$ with the required area shaded.

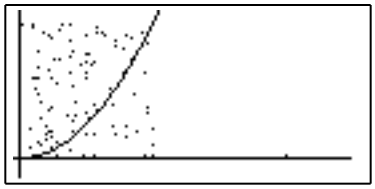


It is clear from the diagram that the area is less than 0.5 square units (half the square) and it also seems to be larger than 0.25 square units (a quarter of the square). In fact, finding this area accurately is very difficult, unless you have some acquaintance with integral calculus. (However, the extraordinary Greek mathematician and scientist Archimedes solved this problem more than twenty centuries ago!)

A Monte Carlo solution to the problem involves generating random points inside the unit square with vertices at the origin, (0,1), (1,1) and (1,0). This square is also shown on the screen above. To generate a random point in the square, we generate a random number for the x-coordinate and then another random number for the y-coordinate. For each random point, we test whether it is in the shaded region (i.e. the y-coordinate is less than the square of the x-coordinate) or whether it is not in the shaded region (i.e. the y-coordinate is greater than the square of the x-coordinate).

After many points are generated, the proportion of the points that is in the shaded area should be about the same proportion that the shaded area is of the area of the square. That proportion is a good estimate of the area of the shaded region.

One way of doing this is to generate a list of 100 random numbers for each of the x-coordinates and the y-coordinates for 100 points and to then plot the points as a scattergraph on the curve, as shown over page:

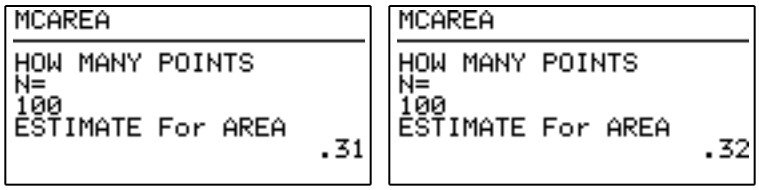


The obvious disadvantage of this process is that it will take a long time to count how many of the points are below the curve and how many are above. In addition, many of the points are so close to the curve itself that you can't decide on which side they fall. Accurate results are unlikely this way, and it will take too long to get a reasonable approximation.

Instead of doing the counting yourself, however, you can use the program MCAREA below, which will do it automatically for you. All you need to decide is how many points to choose.

MCAREA
Print "HOW MANY POINTS
Input N
0 → K:0 → S
Label NEXT
K+1 → K
If random >random^2 Goto MISS
S+1→ S
Label MISS
If K<N Goto NEXT
Print "ESTIMATE For AREA
Print S/N

The screens below show two separate attempts to estimate the area in this way.



In each case, 100 points were chosen. The resulting estimates are close to each other. Of course, you would expect to get a better estimate of the area with more points, although this will take even more calculator time.

Activity 19

1. Use program MCAREA a few times with just 10 points chosen. Why are all the estimates given to just one decimal place? How consistent are the resulting estimates? Compare your observations with your partner's.

2. Why would you expect to get better results from 100 points than from 10 points?

   Use program MCAREA five times with 100 points chosen each time. Why are all the estimates now given to two decimal places? How consistent are the resulting estimates? Compare your observations with your partner's.

3. Try finding the area under the curve by generating more than 100 points, but don't forget that you will drain your batteries if you choose a number that is *too* large.

4. You could change program MCAREA to find the area above the *x*-axis under the curve $y = x^3$ between $x = 0$ and $x = 1$. To do this you need to change one line in

If random >random ^{2} Goto MISS
to
If random >random ^3 Goto MISS

Make this change and use the program to find the area as accurately as you can using a reasonable number of points.

5. The procedure used here would need to be modified to find the area under the graph of $y = 2x^2$ between $x = 0$ and $x = 1$. Draw a diagram to explain why.

Answers to selected questions

Several questions will result in many possible correct answers (abbreviated to MPA).

Activity 1

1. MPA. All numbers will be larger than 0 and less than 1. An assortment of higher and lower numbers should occur.
2. About one tenth of the time.
3. Less than 4 about a third of the time; odd about half the time.
4. MPA, but the results should be fairly even if you play the game for long enough.

Activity 2

1. MPA. Expect about half of the numbers to be less than 0.5..
3. The Fix 2 setting will give all numbers to two decimal places.
4. Numbers are rounded to the nearest integer, which will be zero or one in about equal proportions.
5. MPA. About half of the time the results will be the same and about half of the time they will be different.
6. MPA. About one quarter of the time there will be two heads, about a quarter of the time two tails and about half of the time one of each.

Activity 3

1. All numbers will be larger than 0 and less than 2.
2. All numbers will start with 1, as all will be larger than 1 and less than 2.
3. All numbers will be larger than 2 and less than 6.
4. MPA, e.g. 4*random* + 3 or 2*random* + 5.

Activity 4

2. The results will be the same for numbers larger than zero and different for numbers less than zero. The *int* command is the greatest integer function, while the *ipart* command ignores everything to the right of the decimal point.

Activity 5

2. The results will be the same as all numbers are positive.
3. The commands generate integers in {1, 2, 3, ... ,8} and {3, 4, 5, ..., 8} respectively.
4. *int* (10*random* + 1)
5. Use *int* (12*random* + 1) to generate numbers corresponding to the months.
6. Assign each person a number from 1 to 31. Then use *int* (31*random* + 1) to generate the winning number.
7. Use *int* (45*random* + 1) six times (more if numbers are repeated).
8. MPA, e.g., the first command generates 0's and 1's in about equal proportions; the second generates many more 1's than 0's.

Activity 6

1. MPA. Each time you zoom in or out, a fresh set of random numbers is generated so the graphs will not just be magnified or reduced in size.
2. MPA. The most likely occurrence is that the graph will seem to disappear because of the very large scale.
5. MPA. There will probably be about a quarter of the points in each quarter of the screen.

Activity 7

1. MPA. The table values are recomputed each time, and so *all* of them will probably change.
2. MPA. Expect that the first value will exceed the second about half the time.
3. The calculator does not actually compare the values in the first two columns; rather, it generates a new pair of random numbers. Compare the new pair each time.

Activity 8

1. MPA. It is likely that a double will occur about one sixth of the time, since there are 36 possible pairs of which 6 are doubles.
2. MPA. It is likely that this will occur about one quarter of the time.
3. MPA. A suitable command is {*int (random +0.5),int (random +0.5)*}.
4. MPA. A suitable command is {*int (6random +1),int (6random +1), int (6random +1)*}
5. MPA. A suitable command is {*int (random +0.7),int (random +0.7),int (random +0.7)*}. It is very unlikely that there will be no rain at all on the next three days (represented by {0 0 0}.)

Activity 9

2. MPA. A suitable command is *seq*(*int* (*random* + 0.5),1,100) STO L1.
3. MPA. A suitable command is *seq*(*int* (45*random* + 1),1,6).
4. MPA. (Many people think that runs of three or more heads or tails in a row are too unusual to include.)
5. In order, the commands simulate
   - a single coin toss
   - a set of 10 coin tosses
   - the number of heads in a set of ten coin tosses
   - a set of 8 lots of the number of heads in ten coin tosses
6. MPA. Expect a run of three or more heads about half the time. Expect to get a run of either three or more heads or three or more tails about 80% of the time. Few people are successful at writing down a typical sequence.
7. The command *int* (*random* + 0.85) simulates one throw, while *seq*(*int* (*random* + 0.85),1,5) simulates five throws. Expect her to get five baskets about 40-50% of the time.
9. MPA. This is not a sound position, as runs of three wins or losses happen quite frequently among matched players.

Activity 10

1. The sum of the scores is 728 and the number of scores is 100. So the mean is 728÷100 = 7.28.
2. MPA. Many of the statistics will be similar in size.

Activity 11

1. MPA. The visual representation makes it easier to compare the two distributions directly.
2. MPA. Since the calculator chooses the scales separately for each box plot, they will appear to have the same maximum and minimum scores.
3. MPA. If the *x*-values are not wide enough, parts of the plots may be missing from the screen. If they are too wide, the shape of the box plot may be obscured.
5. MPA. Both distributions are likely to be symmetrical about the median. While minimum scores will be the same, maximum scores will be greater for the 10-sided dice, so the box plot will appear to be more 'stretched'.

Activity 12

1. MPA. Both histograms are likely to be roughly symmetrical with more values in the middle than at the ends. Scores like 6, 7 and 8 are most common, while scores like 2 and 12 are much less common.
2. The *x*-value tells the dice score (the left end of each interval) while the *y*-value tells the frequency. Conventional histograms generally refer to each interval by its mid-point or by a range of values, not just by the left end of the interval.
3. MPA. (Parts of) both will be drawn together, but will not be readily comparable.
6. A suitable command is *seq*(*sum*(*seq*(*int* (*random* +0.5),1,10)),1,500)
7. A suitable command is *seq*(*sum*(*seq*(*int* (*random* +0.25),1,20)),1,50). It is highly unlikely that any students will score 10 or more correct, so this is not a good strategy.

Activity 13

2. MPA. The larger samples are more likely to have means that are close to the whole class (i.e. population) mean, than are the smaller samples.
3. MPA. Even with only 20 people in the room, there will be a matching pair quite often (but less than half the time). With 30 people in the room, there will *usually* be a matching pair.

Activity 14

2. MPA. The proportion of heads is likely to be close to 0.5 in the long run each time.
3. MPA. In the long run, the results should be close to the expected proportions.

Activity 15

2. MPA, but if there are ten tosses per second, a million tosses will take more than a day of non-stop calculation, which is longer than the calculator batteries will last.

Activity 16

1. 6 is the minimum. Theoretically there is no maximum, but it is highly unlikely that more than 30 will be needed and as more than 50 will be needed only about once every ten thousand attempts.
3. MPA. The picture should be fairly clear after about 15 to 20 simulations.

Activity 17

1. MPA. The results will be similar, but much easier to obtain. Expect an average of about 15 cereal boxes to be needed.
2. MPA. Expect about 24 cereal boxes to be needed to get a set of eight cards.
3. MPA. Expect around 30-40 cereal boxes to be needed.

Activity 19

1. All estimates will be fractions out of 10, and so will be numbers to one decimal place. They are unlikely to be very consistent.
2. MPA. With 100 points, there is more likely to be 'long run' effects and results will be more consistent each time. Estimates are now fractions of 100 and thus decimals to two places.
3. MPA. Results will be closer to the exact value of one third with larger samples.
4. MPA. The results will approach one quarter as the number of points sampled increase.
5. A large part of the curve is outside the unit square, unlike the previous two cases. A rectangle of area 2 is involved, instead of a square.