BASIC PROGRAMMING FOR THE CASIO 9850 SERIES OF CALCULATORS

By Martin Schmude

The CASIO 9850 series of calculators offers a very powerful tool, which is the ability to write programs. Although they don't compare in looks to the likes of a home computer or computer game console, they are capable of producing useful and engaging activities for students.

This booklet hopes to teach the basics of programming on the CASIO *CFX9850GB PLUS*. Very little experience is needed to begin this booklet. Over the course of the lessons, you will build a program from scratch and finish with a very useful and dynamic product.

Although it is possible to write a program in the FA-123 program, which can be downloaded for free from the CasioEd website, it is best to learn firstly on the calculator, to ensure correct syntax. The Contents page, that follows, has had the headings linked to the specific parts of the document. This is designed so that when you are viewing it on a computer you are able to jump to a certain lesson.

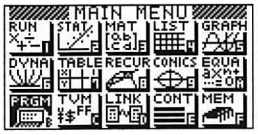
Any feedback or comments, as well as further questions would be very welcomed. I hope you enjoy the learning experience.

Marty Schmude
mschmude@joeys.org

CONTENTS

LESSON 1 – Starting a new program

1. Once your calculator is turned on, enter the PRGM mode, using either the arrows on your keypad and press the blue **EXE** key, or simply press the button which corresponds to the letter B (log).



2. Most likely, the program list is empty. If there are programs present, just ignore them for the moment. To start a new program, press NEW (F3).

3. Enter in a name for your program. For starters, we'll call our new program "TEXT". To enter in the letters, use the buttons that correspond to the letters above the keys. Once the word "TEXT" is written, press the blue **EXE** key to enter it. You will be moved then to the working area of the TEXT program. This is where the program writing begins.

LESSON 2 – Displaying text

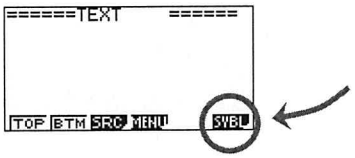
Most programs require text to be displayed. In this lesson we will write some text and run the program to view it.

1. If you haven't already, enter the TEXT program by entering PRGM mode from the main menu and then place the cursor on TEXT and press EDIT (F2).

2. Let's enter in the text "FIRST NUMBER".

   Notice that the text is in inverted commas. This is intentional because when you enter text, which is to be displayed, you must encase it in inverted commas.

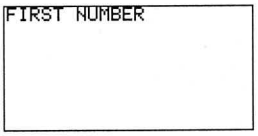
   The " is found in the SYBL option (F6), located in the bottom right hand side of the screen. In this menu is the " option (F2).



3. Instead of pressing the **ALPHA** key each time you enter a letter, press the yellow SHIFT button, followed by the **ALPHA** key. This puts A-LOCK on (a flashing capital A is shown), which allows you to enter the letters, one after the other. To enter a space, press the period (.) key. Notice the red word **SPACE** above the period button (.). This tells you that you must press **ALPHA** to enter a space, unless A-LOCK is on.

4. Once the words are entered, press " (F2) again and then **EXE**. Pressing **EXE** at the end takes you to a new line, hence the ↵ symbol.

5. Now let's test it. Press EXIT twice (once to exit the SYBL menu and the other to exit the program), then press **EXE** or F1 to run the TEXT program.



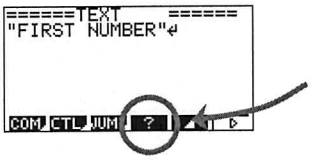
LESSON 3 – Prompting for user input

We are going to use the text that you entered in the previous lesson as a prompt for the user to enter a value. You will be introduced to the PRGM menu.

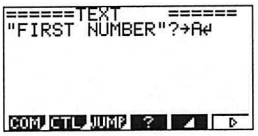
1. Enter the TEXT program by entering PRGM mode from the main menu and then place the cursor on TEXT and press EDIT (F2).

2. Place the cursor after the last " symbol, i.e. on top of the ↵, by using the arrows on the keypad. If you were to enter something now, the calculator would overtype anything in its way. To avoid this, there is an insert option (INS). Press SHIFT **DEL** and a flashing box will appear.

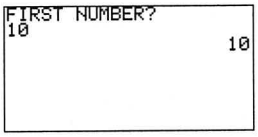
3. We will enter the ? symbol, which tells the calculator to stop and wait for a value to be entered. This symbol is found in the PRGM menu, which is located above the VARS button in the top row. Since the word PRGM is in yellow, it means you must press SHIFT before pressing VARS, in order to access it.



4. Enter the ? symbol by pressing F4. The calculator now requires a place to store this input. The calculator has 26 memories; each corresponds to a letter in the alphabet. Let's assign this value to the variable A. After entering ?, press the → button on your keypad followed by the letter A (**ALPHA** A). It should look like the figure below.



5. Let's test it. Exit the program and run it by pressing **EXE**. The program should stop and wait for you to enter a value by pressing the numbers, then hitting **EXE**. Since there is no more code to read, the calculator displays the entered value.



How it works

Entering the ? symbol tells the calculator to pause while the user enters a number. The →A then tells the calculator to store the value with the variable A. This storing is essential to programs, so that we can recall this data to perform other tasks, which will be our next lesson.

LESSON 4 – Using input data

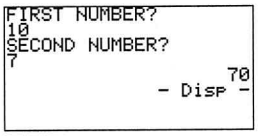
This lesson we will learn the basics of manipulating data that has been entered by the user.

1. Enter the PRGM mode and enter the TEXT program by pressing EDIT (F2)

2. Place the cursor on the second line in the program window. We are going to enter the line "SECOND NUMBER"?→B↵. Remember the " symbol is in the SYBL menu, the ? symbol is in the PRGM menu, you can use A-LOCK, and press **EXE** at the end of the line. The result should look like the figure below.

3. Let's use the two numbers that the user entered, which are now stored as the variables A and B, and then display the product of them. Place the cursor on the third line by pressing the down arrow on the keypad. Now enter the line A×B◢. The ◢ symbol is in the PRGM (SHIFT VARS) menu and it means 'stop the program and display this result'.

4. Let's see it in action! Exit the program and run it. Enter the first number to be 10 and the second number to be 7. It should look like the figure below. Notice the word – Disp – after the result. This means 'display' and to continue, press **EXE**.



How it works

The calculator prompts the user to enter two values, which are then stored as the letter A and B using the ?→ symbols. The next command then is to multiply these two numbers and display the result using the ◢ command.

Why Not Try...

1. Try writing a new program called "SUBTRACT", which finds the difference between two numbers and displays the answer.
2. Write another program that asks for three inputs from the user, then adds them together and displays the result.

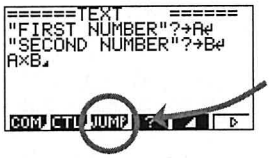
LESSON 5 – Introducing a Goto jump command

Most programs involve loops or 'If' statements. Loops are when the program restarts from a certain point, perhaps with some different parameters. An 'If' statement is a way of checking the condition of a variable. Both loops and 'If' statements benefit from jump commands, that is, when the program is told to 'jump' to a different part of the program. We will introduce a type of jump statement, called a Goto command.

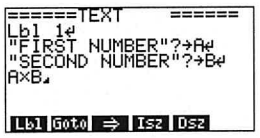
1. If you haven't already, enter the TEXT program by entering PRGM mode from the main menu and then place the cursor on TEXT and press EDIT (F2).

2. In order for the jump command to work, the program needs to know where to go. This means the first step is to insert a 'label'. Place the cursor at the very beginning of the text program. Press SHIFT DEL to change the cursor to insert (INS).

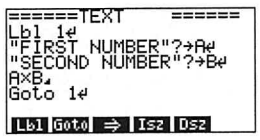
3. Enter the PRGM menu by pressing SHIFT VARS. You will notice the JUMP menu. Press F3 to enter this menu.



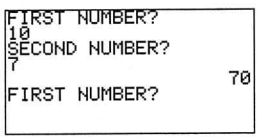
4. There is an option called Lbl. Select this command by pressing F1. You need to then name the label. We will call it Lbl 1. Once complete, enter a new line by pressing **EXE**.



5. Move the cursor to the bottom of the program by pressing the down arrow repeatedly. While the JUMP menu is still open, you will notice the Goto command. Enter this operation by pressing F4. Entering the command is not enough though. We need to tell it where to go. We want the program to go to Lbl 1, so add 1 after the Goto command so it reads Goto 1.



6. Test it! Exit the program and run TEXT by pressing **EXE**. Nothing changes in the output except that after the word – Disp – is displayed and **EXE** is pressed, the program immediately jumps back to Lbl 1, and starts all over again.



How It Works

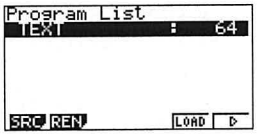
When you enter a `Goto` command, it must have a label, `Lbl`, to jump to. These labels can be named with numbers or letters, but numbers are easier to begin with.

LESSON 6 – Renaming a program

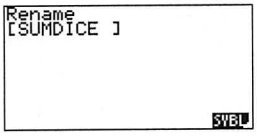
Sometimes it is necessary to rename a program. We are going to change the name of the TEXT program to SUMDICE.

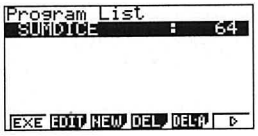
1. Enter PRGM mode from the main menu. Place the cursor, using the arrow keys, on the program TEXT, but do not press EDIT.

2. The option to rename the program is not seen on the screen and is not hidden in the keypad anywhere. Notice in the bottom right hand side of the screen, an option which looks like ▶, nicknamed 'Des' by Anthony Harradine. This symbol means the bottom menu continues. Press F6 and you'll notice 'des more options.



3. Notice the option REN, which stands for 'rename'. Press F2 and you'll be prompted to change the name of the program. Enter the word SUMDICE. Notice A-LOCK is already on. Press **EXE** when complete. The program is now called SUMDICE.



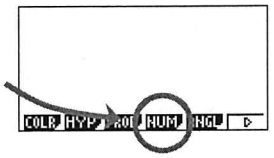


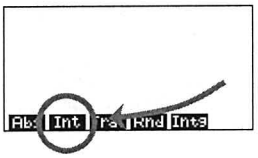
LESSON 7 – Understanding the Int function

The Int function, which stands for integer, on the calculator performs a very important command. It can be used for such things as checking the value of a user's input and the simulation of dices. We are going to use it the latter reason.

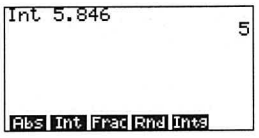
1. From the main menu, enter RUN mode.

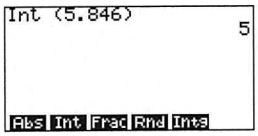
2. The INT function takes the integer part of a number. It doesn't round the number up or down, it just simply truncates the decimal places. Let's experiment with some numbers. The Int function is in the OPTN menu. Press OPTN, which is next to the SHIFT button. Follow this with 'Des' (F6) to find the NUM (F4) option.





3. Press Int (F4) and enter the number 5.846, then press **EXE**. You'll notice that the function simply ignores the decimal numbers. Also note that it is not unusual to place the number in parentheses.





Why Not Try...

1. Experiement with other numbers so you feel quite comfortable with this function.

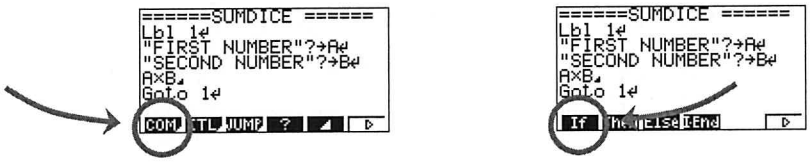
LESSON 8 – Enter an If statement

If statements are one of the most popular commands in programming. They offer enormous control and flexibility, and play an essential role. In this lesson, we are going to restrict the values the user can enter when prompted for the FIRST NUMBER and the SECOND NUMBER.

1. If you are not yet in SUMDICE, enter it by pressing the EDIT option in PRGM mode.

2. You are going to limit the user to entering in an integer value. You'll be shown now how adjust the FIRST NUMBER and you can do the same thing to the SECOND NUMBER. Place the cursor on the first inverted comma of "SECOND NUMBER. Select the INS (insert) option by pressing SHIFT **DEL**.

3. Enter the PRGM menu by pressing SHIFT VARS. You will see a menu labelled COM (F1), which stands for 'commands'. In this menu is the If statement.



4. We are going to test A and make sure it is a whole number. If it is not, we will prompt them to enter FIRST NUMBER again. Set the program to test the user's input for the FIRST NUMBER, which is variable A.

   An If statement must always be accompanied with a Then statement, and finished with an IfEnd command.

   Enter in the highlighted text below. Remember to press **EXE** at the end of each line, Then and IfEnd are found in the same menu as the If command. Goto is in the PRGM → JUMP menu and Int is under the OPTN → NUM menu. One symbol we haven't seen yet is ≠. It is found in the PRGM menu, then 'Des', followed by the REL (relations) menu.

```
======SUMDICE ======
Lbl 1↵
"FIRST NUMBER"?→A↵
If Int (A)≠A↵
Then Goto 1↵
IfEnd↵
"SECOND NUMBER"?→B↵
A×B↵
Goto 1↵
```

5. Let's see it work. Press EXIT until you are in the program list, then press **EXE** to start. When prompted for the FIRST NUMBER, enter a non-integer value, like π. You should be prompted again for the FIRST NUMBER.

How It Works

We have told the program to take the integer of A and test it against the actual value of A. If these two don't equal, then obviously A is not an integer. The next part of the statement is the Then command, since we said, 'if they don't equal, then go to label 1'. If A is an integer, the program will ignore the Then statement.

Why Not Try...

1. Enter an If statement to test the SECOND NUMBER. The program will be exactly the same except it will be letter B.

LESSON 9 – Entering text and a `Clrtext` command, and adjusting a `Goto` command

*It is assumed that you have completed the **Why Not Try...** section at the end of the last lesson. If you haven't, go back there now and have a go.*

At the moment, our SUMDICE program has some user-friendly issues. If the user enters a non-integer value, the program jumps back to the start. There are four main problems with this:

1. The user doesn't know why the program has restarted,
2. If they are up to the SECOND NUMBER, they have to enter the FIRST NUMBER again.
3. The text is not cleared when the program restarts, which is only cosmetic but messy nonetheless.
4. The user doesn't know what the final answer is a result of.

At the moment, your program should look like this:

```
======SUMDICE ======
Lbl 1↵
"FIRST NUMBER"?→A↵
If Int (A)≠A↵
Then Goto 1↵
IfEnd↵
"SECOND NUMBER"?→B↵
If Int (B)≠B↵
Then Goto 1↵
IfEnd↵
A×B◢
Goto 1↵
```

Adjusting a `Goto` command

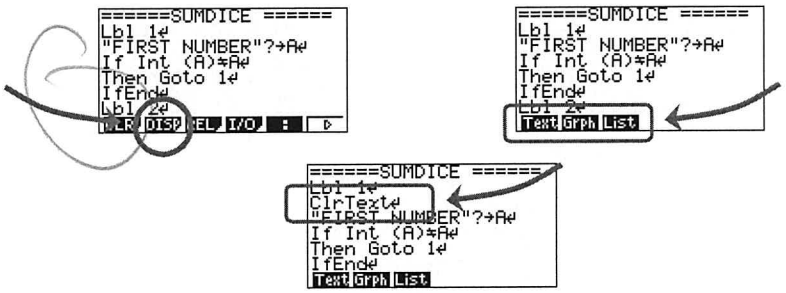
1. Let's start with the easiest one first, point 2: having to start again when prompted for the SECOND NUMBER. Instead of jumping to Lbl 1, let's insert a Lbl 2 just above the line "SECOND NUMBER"?→B↵. Of course this means we will have the change the `Goto` command to 2, instead of 1. Remember to use INS (insert). Go ahead and enter that now. The result should look like below.

```
======SUMDICE ======
Lbl 1↵
"FIRST NUMBER"?→A↵
If Int (A)≠A↵
Then Goto 1↵
IfEnd↵
Lbl 2↵
"SECOND NUMBER"?→B↵
If Int (B)≠B↵
Then Goto 2↵
IfEnd↵
A×B◢
Goto 1↵
```

2. Exit the program and give it a try. Test the second input with a non-integer value. The program should only prompt you for the SECOND NUMBER.

Clearing text

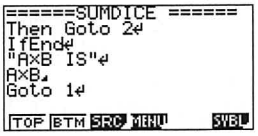
3. The next part is clearing the text, both at the start and if the program has to restart. This can be done with the same well-placed command. There is an option in the PRGM (SHIFT VARS) menu. We are looking for the CLR (F1) menu. You will have to press 'Des' (F6) to cycle to it. In this menu, you will find all the commands for clearing, like text or graphs. Place your cursor on the first inverted comma in the line "FIRST NUMBER and change the cursor to insert INS (SHIFT **DEL**). Press Text (F1) and you will a see a ClrText appear. Press **EXE** now to enter a new line.



4. Enter in a ClrText command after Lbl 2. Place the cursor on the first inverted comma in the line "SECOND NUMBER, set the cursor to INS and press Text (F1) again. Exit SUMDICE and give it a try. You will notice the text clears each time the program restarts.

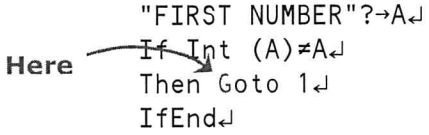
Entering text and entering text in an If statement

5. We are going to adjust two things in this part. Firstly, letting the user know where the result comes from. Move the cursor so it is over the A in the second last line, A×B◢. Change the cursor to INS (insert) and add the text "A×B IS", remembering to press **EXE** to start a new line. Figure below might help.



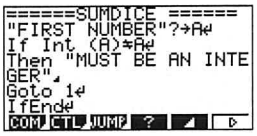
6. Try it now.

7. The second and last part of the cleanup is telling the user why their input was not accepted if a non-integer number was entered. We are going to tell the user *before* the jump. Position the cursor on the G of the first Goto in the program (5th line) and set it to INS.



Enter the text, "MUST BE AN INTEGER"◢. We haven't yet had text that has been followed by the 'display' symbol, ◢. We use it here because if we did not,

the program would not stop, and the user would never really see it. Pressing **EXE** continues the program, which is the jump back to Lbl 1. Notice that when the display command was entered, it jumped to a new line.



8. Do the same for the SECOND NUMBER. Place the cursor on the G on the second Goto command (12th line now, after the above adjustment)

```
======SUMDICE ======
Lbl 1
ClrText
"FIRST NUMBER"?→A
If Int (A)≠A
Then "MUST BE AN INTE
GER"
Goto 1
IfEnd
Lbl 2
ClrText
"SECOND NUMBER"?→B
If Int (B)≠B
Then "MUST BE AN INTE
GER"
Goto 2
IfEnd
"A×B IS"
A×B
Goto 1
```

9. Let's try run SUMDICE now and make sure it all works. Experiment with different values.

LESSON 10 – Adding other conditions in an `If` statement

At the moment we have restricted the user to entering whole numbers, if they wish to proceed with the program. In this lesson, we will go further than that and restrict the user to entering numbers from 1 to 10. To do this, we will use some logic statements.

1. Enter the SUMICE program.

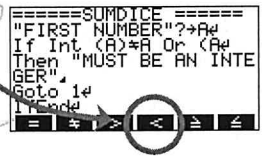
2. Place the cursor at the end of the line of the first `If` statement.

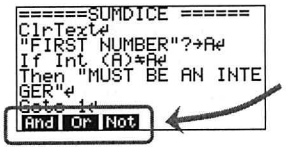
```
"FIRST NUMBER"?→A↵
If Int (A)≠A↵
Then "MUST BE AN INTE
GER"◢
```

Here

3. We are going to add that A must be greater than or equal to 1 OR less than or equal to 10, shown below. This condition, mixed with the integer check, will guarantee that only integers entered from 1 to 10 will miss the 'jump to the start' command, `Goto 1`.

   Notice the word OR in the previous paragraph. This is the logic gate that we are going to use. It is hidden under OPTN (next to SHIFT), then the LOGIC menu is found by pressing 'Des' twice and it will correspond with F4. In this LOGIC menu, you will see the three most basic logic gates, AND, OR and NOT.



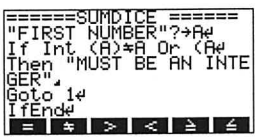



```
"FIRST NUMBER"?→A↵
If Int (A)≠A Or (A<1)
 Or (A>10)↵
Then "MUST BE AN INTE
GER"◢
```

   The brackets around the conditions are just to help us visually about what is going on.

4. Set your cursor to INS and press OR (F2). This inserts an OR statement. Type in the bracket and then A. We need to find the < symbol.

5. Now enter the PRGM menu (SHIFT VARS) and find the REL menu by pressing 'Des'. Remember we found the ≠ symbol in the REL (relations) menu. The inequality signs are in there as well.



6. Enter the rest of the `If` line shown above.

7. Let's test it. Exit the program and press **EXE** to run it. You'll notice one problem when we execute the program and enter an integer outside our specified range. I'll leave you to see it what the user-friendly problem is.

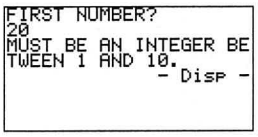
Why Not Try...

1. Make the same adjustment to the SECOND NUMBER section of the SUMDICE program. Instead of A the variable will be the letter B.
2. Update your SUBTRACT program with some of the new features you've learned. Perhaps add a condition that makes the user choose a FIRST NUMBER that is less than the SECOND NUMBER, or restarts the program if the answer is less than zero.

LESSON 11 – Adding more text in an If statement

If you didn't pick up on the problem at the end of the last lesson, it was that the user did not know why their number wasn't accepted, even though it was an integer. It is because our If statement restricts their input, but the user doesn't know all the restrictions. What we need to add is some more text to let the user know that if they enter a value that is not a whole number, or less than 1 or greater than 10, the program won't proceed.

1. If we were to directly add on to the line "MUST BE AN INTEGER BETWEEN 1 AND 10.", the displayed result would look like the figure below.



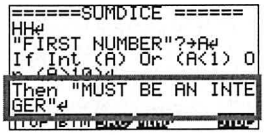
   Notice how the line breaks into the next line, which at best looks bad, and at worst is difficult to read.

   If you're not already in the SUMDICE program, enter it now.

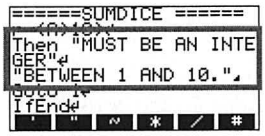
2. You are going to cut the line into two pieces, so that all the words appear as a whole. The best place to stop the first line would be after the word INTEGER. That is so we use as much of the line as possible. Place the cursor on the ◢ after the word INTEGER and set the cursor to INS.

```
"FIRST NUMBER"?→A↵
If Int (A)≠A Or (A<1)
 Or (A>10)↵
Then "MUST BE AN INTE
GER"◢
```
Here

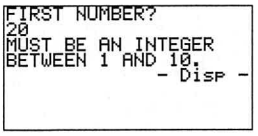
3. Press the grey **DEL** button next to the **AC/ON** button to delete the symbol ◢. If you just press **EXE** to start a new line, it would not remove this symbol, and we don't want to stop the message midway through it. Once this is done, press **EXE** to enter a new line. The symbol after INTEGER should be ↵.



4. The next part is to add the new line. Enter the text "BETWEEN 1 AND 10."◢. Use the decimal point as the full stop. Notice how once again we use the 'display' command because we want the two lines to be shown then the program to pause, while the user reads the message.



5. Let's see it in action. Exit the program and run SUMDICE. When prompted for the FIRST NUMBER, enter a number like 20, and the result should be as below.



When **EXE** is pressed, the text should be cleared and you should be prompted for the FIRST NUMBER again. It should also be mentioned that if the user got to the SECOND NUMBER, and entered an 'illegal' value, they would be prompted for the SECOND NUMBER again, but the FIRST NUMBER will still be present and unchanged.

Why Not Try...

1. Do what you've just done to the FIRST NUMBER Then statement, to the SECOND NUMBER Then statement. Just so you can check if you have everything correct, below is the entire script for the SUMDICE program so far, as it looks on the calculator.

```
======SUMDICE ======
Lbl 1↵
ClrText↵
"FIRST NUMBER"?→A↵
If Int (A)≠A Or (A<1)
 Or (A>10)↵
Then "MUST BE AN INTE
GER"↵
"BETWEEN 1 AND 10."◢
Goto 1↵
IfEnd↵
Lbl 2↵
ClrText↵
"SECOND NUMBER"?→B↵
If Int (B)≠B Or (B<1)
 Or (B>10)↵
Then "MUST BE AN INTE
GER"↵
"BETWEEN 1 AND 10."◢
Goto 2↵
IfEnd↵
"A×B IS"↵
A×B◢
Goto 1↵
```

If this is how your program looks, then you should be very proud of yourself!

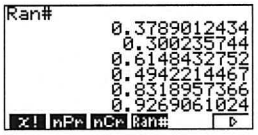
LESSON 12 – Understanding the Ran# function

One of the most useful parts of the calculator is the ability to generate pseudo-random numbers. If you would like to know more about them, go to How Stuff Works website search for 'pseudo-random numbers'. This lesson will go over understanding the simulation capabilities of the calculator.

1. Enter RUN mode from the main menu.

2. The Ran# command is in the PROB menu, and the PROB menu is found in the OPTN menu. Press OPTN, then 'Des' (F6) until you see the PROB menu (F3). Now enter the PROB menu.

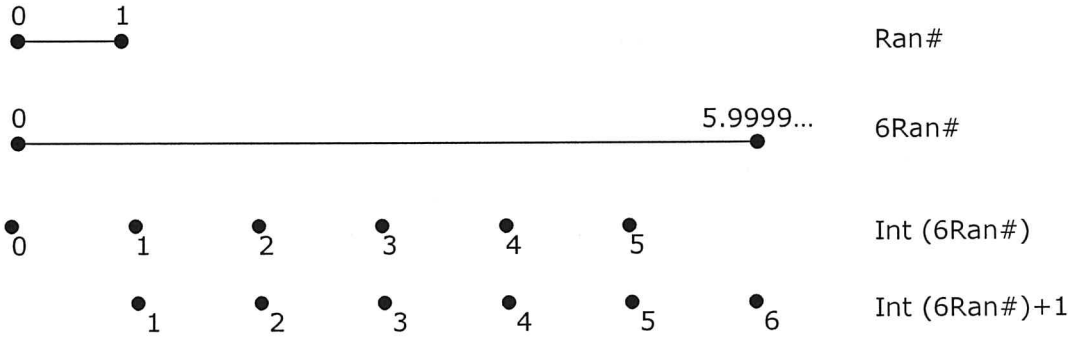
3. Press Ran# (F4) and then **EXE** repeatedly. You will see the calculator performs the same action each time the **EXE** button is pressed. You could be *almost certain* that you will not have the same numbers that are shown below.



You may have noticed that the numbers generated lie between 0 and 0.9999...

4. Our goal for this lesson is to generate randomly the numbers of a die, 1 to 6. If we multiply Ran# by 6, this will produce numbers 0 to 5.9999... Now comes the tricky bit – we use the `Int` function that we learnt about earlier.

   If we integerise the results of 6Ran#, we will generate numbers from 0 to 5. So then the last step then is to add 1 to the result, to make the numbers from 1 to 6. The diagram below might help.



5. To enter this in the calculator, all the functions are in the OPTN menu. The Int (F2) function is in NUM (F4) and Ran# (F4) is in PROB (F3).

Why Not Try...

1. While you are still in the RUN mode, see if you can make a formula to generate random numbers between:

   (a) 0 and 100
   (b) 1 and 36 (good to use in class for bingo)
   (c) 20 and 50
   (d) −10 and 10

LESSON 13 – Introducing a For statement

A For statement is a very efficient way of performing loops a certain number of times. We are going to use this to simulate the summation of two dice, hence the program name SUMDICE. The first few steps are to adjust the program before we introduce the For statement.

1. Enter the SUMDICE program.

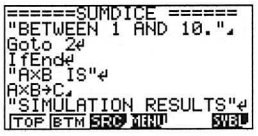
2. Delete the last line of the program, Goto 1, using the **DEL** button. We want to add more to it now and do not wish for it to start again.

3. The final value at the end of the program, which is A×B, is not yet saved on the calculator. We are going to assign this value to the variable C, and then use it in later calculations. After the A×B, insert the symbols →C.

```
IfEnd↵
"A×B IS"↵
A×B→C◢
```

4. Number C is going to be the number of simulations the calculator will perform. Go to the very bottom of the program. There is a shortcut way of doing this, which is to press the BTM (F2) button found in the opening menu.

5. Enter the text "SIMULATION RESULTS"↵ on the last line, as shown below.



6. Now for the For statement. The syntax for a For statement will need to be explained but is simple. Below is perhaps the simplest For command you could have.

For Statement

```
1 For 1→Z to 10↵
2 Z◢
3 Next↵
```

For Statement explained...

- **Line 1**: The variable Z is set to 1.
- **Line 2**: The value of Z is displayed (1).
- **Line 3**: The program jumps back to the For command because it has reached the word Next.
- **Line 1**: 1 is then added to Z, making its value 2.
- This continues until Z equals 10, when it does it the last time.

Below are the lines that need to be entered into our SUMDICE program. It will include the initial For statement, followed by the simulation of the two dice. After this, the calculator will then compute the summation. It will repeat this procedure until it has run through it C times.

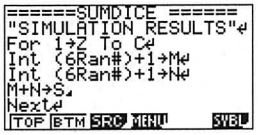
```
For 1→Z To C↵
Int (6RAN#)+1→M↵
Int (6RAN#)+1→N↵
M+N→S◢
Next↵
```

The highlighted portions are the bits that have to do with the For statement. In the first line, we specify that the calculator is to start counting at 1 and is to use Z as the variable that the calculator uses to count how loops it has done. The Next tells the calculator to start again from the For statement, and the value of Z is increased by 1 automatically. The increments can be adjusted but this will not be discussed here now.

In the other lines, you will notice the two simulations with the Ran# and Int functions, whose values are stored as M and N respectively. The following line then adds the outcomes together, stores the result with the letter S and displays the result, shown by the ◢ symbol.

7. Enter the PRGM (SHIFT VARS) menu and then the COM (F1) menu, which is where we found the If commands. You will need to press 'Des' to find the For, To and Next commands.

8. Enter in the complete For statement as shown in point 6. It should appear as in the figure below.



9. Give it a rip! Exit the program and press **EXE** to run it. Set the FIRST NUMBER to 5 and the SECOND NUMBER to 2. This will produce 10 randomly generated summations. You will notice that 11 numbers appear because the last number is displayed again as the overall final result.

How It Works

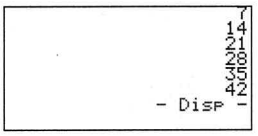
A For statement creates a way of looping a certain number of times. This is useful, especially when running simulations many times, or drawing lines when doing graphics, which will be discussed in Section 2 of this booklet. The key to using a For statement is to ensure 4 main things:

- Have a number the calculator starts counting from.
- Include a variable that calculator uses to count with.
- Have a number that the looping finishes at.

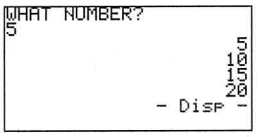
- A `For` statement is always accompanied with a `Next` command, just like an `If` statement must be accompanied by a `Then` and `IfEnd` statement.

Why Not Try...

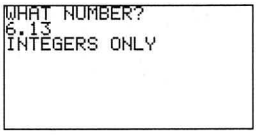
1. Write a new program called `TIMES` that uses a For statement so to display the 7 times tables. Make sure you clear the text at the start.

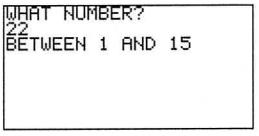


2. Using the `TIMES` program you just wrote, why not add a feature that allows the user to enter the number who's times table will be displayed.



3. Using the `TIMES` program, add a condition to this 'input feature', so that it restricts the user to integers and numbers between 0 and 15.





LESSON 14 – Storing data in a list

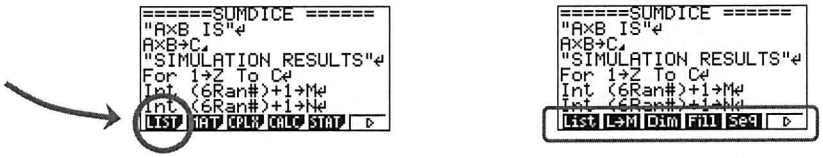
The usefulness of the results in the simulation is limited because the outcomes are not stored anywhere, since the variable S is always changing during each loop. This is why it is a good idea to store data in lists (sometimes called arrays), where it can then be analysed later. This is the purpose of this lesson.

1. Enter SUMDICE program and place the cursor on the first inverted comma of the line "SIMULATION RESULTS.

2. Set the cursor to INS (insert).

3. In order to store the data in a list, we must first set the dimensions of the list. This means telling the calculator how many elements there are to be stored. Since we are using the variable C as the number of simulations, this will be used as the dimensions of list 1.

   The line will read C→Dim List 1. The commands Dim and List are found in the LIST menu, which is in the OPTN menu. Press OPTN, followed by LIST (F1). Just about any action that relates to lists will be found in this menu.



A Bit of Extra Help

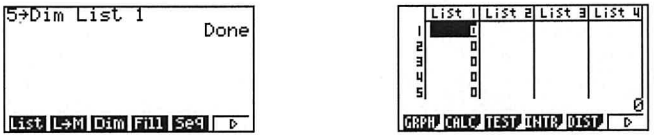
Just to help you understand what has just happened, feel free to try this little exercise.

Enter RUN *mode from the* Main Menu. *Enter this line and press* **EXE**.

5→Dim List 1↵

This line says that I want to have 5 'empty' elements in List 1 (they are in fact filled with the value of 0).

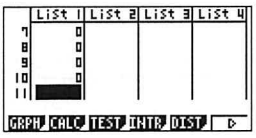
Press MENU *and then enter* STAT *mode. You will see that there are 5 'empty' elements in List 1.*



4. Go ahead and enter the line C→Dim List 1↵. Remember to press **EXE** at the end to start a new line. The product should look like this:

```
A×B→C
C→Dim List 1
"SIMULATION RESULTS"
```

5. Before we move on to the next part, let's see what the effect of the new line makes. Exit the program and run it by pressing **EXE**. Arrange it so that C works out to be 10. When the text SIMULATION RESULTS is displayed, press your grey **MENU** key, then 2, to enter STAT mode.



You will notice that this command has created 10 'empty' elements in list 1. As a rule, whenever you want data to be stored in a list (during the execution of a program), you must first create empty elements using the command described above. This is, if you tried to enter data in position 11, you would receive an error message.

6. Return to the main menu by pressing **MENU** and then enter PRGM mode, and then SUMDICE.

7. Now move the cursor to the beginning of the final line, which is the Next↵ command.

```
Int (6Ran#)+1→M↵
Int (6Ran#)+1→N↵
M+N→S◢
Next↵
```

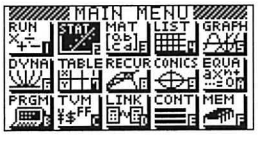
Here →

8. Set the cursor to INS (insert).

9. We are going to add a line that enters the result of the summation, S, in list 1 of STAT mode. This line will be added within the For statement, in order to make use of the incremental qualities of the variable Z. The rows that the results will be entered in will be from 1 to C. Below is the line to be added. The square brackets are the ones above the + and – buttons. They are in yellow so the **SHIFT** button must be pressed to access them. A deeper explanation of this step is at the end of this lesson.

```
M+N→S◢
S→List 1[Z]↵
Next↵
```

10. On the line below the command Next↵, add the text "PRESS MENU 2"↵. This tells the user how to quickly enter STAT mode. Pressing MENU will generally take you from anywhere in the calculator to the main menu. The number 2 is because STAT mode is the second option in the main menu, as shown below.



11. Let's test it now. Exit the program and press **EXE** to start it. Enter the FIRST NUMBER as 5 and the SECOND NUMBER as 2. Once the program is completed and the message, PRESS **MENU** 2 has been shown, press **MENU** then 2 and scroll through the results stored in list 1.

How It Works

Perhaps the best way to explain this lesson is to look at the coding for the entire For statement.

```
For 1→Z To C↵
Int (6Ran#)+1→M↵
Int (6Ran#)+1→N↵
M+N→S◢
S→List 1[Z]↵
Next↵
```

When the program first comes to the For statement, it will set Z to 1 and go through and execute everything on the way to the Next command. This means it will get to S→List 1[Z] and send the value of S (which has been set as M+N) to List 1, row Z, which at the moment is 1. After this, the program will move on to the next line, which is the Next command in this case.

Once it gets to the Next command, the program will go back to the For command and do it again, except Z will now be set to 2. So then the new value of S will be sent to List 1, row Z, which is now 2. This will continue over and over again until Z=C.

Notice how having the S→List 1[Z] command within the For statement utilises the way Z increases by 1 each time.

Just as a side note, the maximum number of elements that this particular calculator can handle in 255.

Why Not Try...

1. Write a new program that will generate an integer between 1 and 20, which is then used for determining the number of loops in a For statement. The purpose of each loop will be to display the number of loops that have occurred up till then.

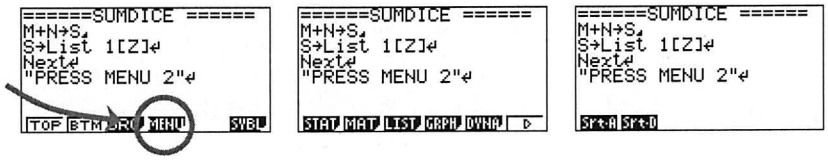
LESSON 15 – Sorting a list

It can make things much easier to analyse a list if the data is sorted in an order, whether it be ascending or descending order. The calculator is capable of doing both. In this lesson, we are going to learn how to sort a list.

1. Enter the SUMDICE program and press BTM (F2), which sends the cursor to the very bottom of our code. We are going to place the sort command here so it will sort the list before the user is told to go STAT mode.

2. Set the cursor to INS (insert).

3. All we need to find is the Sort command. The menu, which appears when you first enter a program, contains the MENU (F4) menu. In this you will find a LIST (F3) menu. Press this option and you will see a Srt-A (F1) and Srt-D (F2) option. Srt-A means 'sort ascending' and Srt-D means 'sort descending'.



4. Press Srt-A (F1). It will appear SortA(, like the figure below shows.

5. Enter the command List 1. Recall that the way to get to the command List is through OPTN, then LIST (F1) and then LIST (F1). Type in 1 after that, close the brackets and press **EXE** so it is on its own line. The final result should read:

```
Next↵
SortA(List 1)↵
"PRESS MENU 2"↵
```

6. Exit SUMDICE and try it. When you enter STAT mode, List 1 will be sorted in ascending order.

LESSON 16 – Eliminate display command and enter text

Since the results are now being stored in a list, it seems unnecessary to have to view them in RUN mode, where the program is executed in. This is very time-consuming, especially if the number of simulations is large. In this lesson, we are going to eliminate the display command, ◢, and add some extra text.

1. Enter the SUMDICE program.

2. We are going to delete and replace the command that displays each result as it happens. Place your cursor on the display command, in the For statement, and set it to INS (insert).

```
For 1→Z To C↵
Int (6Ran#)+1→M↵
Int (6Ran#)+1→N↵
M+N→S◢
S→List 1[Z]↵
Next↵
```
Here

3. Press **DEL** next to the **AC/ON** button to delete this command. Replace it with a new line symbol, ↵. This will stop displaying of each result as the program proceeds.

```
For 1→Z To C↵
Int (6Ran#)+1→M↵
Int (6Ran#)+1→N↵
M+N→S↵
S→List 1[Z]↵
Next↵
```
Here

4. What would be a good idea now would be to enter the word "COMPLETE" at the end of the program, just before the "PRESS MENU 2". This will let the user know that the SIMULATION RESULTS are complete.

```
For 1→Z To C↵
Int (6Ran#)+1→M↵
Int (6Ran#)+1→N↵
M+N→S↵
S→List 1[Z]↵
Next↵
"COMPLETE"↵
"PRESS MENU 2"↵
```

5. This should do it. Exit SUMDICE and give it a try. Since the program will run much more quickly now, why not set A×B to be 100. The final result before going into STAT mode should look like the figure below.

```
SECOND NUMBER?
10
A×B IS
                100
SIMULATION RESULTS
COMPLETE
PRESS MENU 2
```

The Final Program

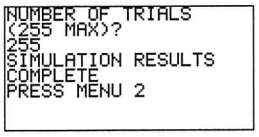
Well done, you've made to the end of the first section! You have learnt a great deal considering the first lesson was learning just how to display text. Now you have some idea of the basics of writing a program, and understand the power of some commands like an `If` statement. Well done!

Below is the complete script of the SUMDICE code that you have written over the lessons. Our next section is where the real fun begins – graphics – coming soon!

```
======SUMDICE ======
Lbl 1↵
ClrText↵
"FIRST NUMBER"?→A↵
If Int (A)≠A Or (A<1)
 Or (A>10)↵
Then "MUST BE AN INTE
GER"↵
"BETWEEN 1 AND 10."◢
Goto 1↵
IfEnd↵
Lbl 2↵
ClrText↵
"SECOND NUMBER"?→B↵
If Int (B)≠B Or (B<1)
 Or (B>10)↵
Then "MUST BE AN INTE
GER"↵
"BETWEEN 1 AND 10."◢
Goto 2↵
IfEnd↵
"A×B IS"↵
A×B→C◢
C→Dim List 1↵
"SIMULATION RESULTS"↵
For 1→Z To C↵
Int (6Ran#)+1→M↵
Int (6Ran#)+1→N↵
M+N→S↵
S→List 1[Z]↵
Next↵
SortA(List 1)↵
"COMPLETE"↵
"PRESS MENU 2"↵
```

Why Not Try...

1. It seems silly that the number of trials is the result of the product of two numbers. Why not rewrite the beginning of SUMDICE so that the user is prompted at the beginning for the NUMBER OF TRIALS to enter in how many simulations there should be. The maximum number of entries in list is 255.



2. Since the maximum number of trials is 255, set up the program so that the user must enter an integer value which is 255 or less.

3. SUMDICE is a very useful program when talking about probability, normal distribution and statistics. Experiment with the program and take note of the mathematics it is doing, like displaying the data stored in List 1.

HAVE FUN PROGRAMMING