

Mental Maths



Teacher Notes & Answers

7 8 9 10 11 12



TI-Nspire



Coding



Student



120 min

Teacher Notes:

A PowerPoint slide show is provided to help run an introductory game for the entire class. If a projector is not available, generating random numbers and writing them on the board will also suffice.

After playing a game, students write a program to simulate the game. The idea of students simulating the game is for them to practice prior to the next time the game is played in class. In addition to being an introductory coding activity, playing the game has many benefits:

- Improves numerical estimation skills;
- Problem solving;
- Order of operations, including grouping (parenthesis).

There are numerous opportunities for students to elaborate on their coding:

- Include count-down notifications “20 seconds to go ...” or a count-down timer;
- Include error trapping to ensure the user enters only 3, 4 or 5 small numbers;
- Introduce variables such as the option to change the time or quantity of numbers;
- Automatic scoring (enter your expression to receive a score) and the potential to retain scores;
- Equivalent games (Seeding the random variable generator means students can receive the same number set)

Going beyond the basic game, there are numerous opportunities to extend the activity into a full investigation.

- **How many small numbers should you choose?**
Explore whether it is better to have 3, 4 or 5 small numbers. Which one generally results in a higher score.
- **Would you like to buy some time?**
The idea here is that students may be willing to sacrifice points for extra time. To get students thinking about how much the extra time might be worth, run a reverse auction. Start the bidding at 20 points for an additional 10 seconds of time. The points are subtracted from your score at the end of a game. If there are no bids for 20 points, “do I hear 19 points ... what about 18 ... ?” until a student is happy to sacrifice the number of points from their final score. This time when the clock stops, those students have the extra time allocated, then subtract their bid from the final score. Bidding must take place before the numbers are generated!
There is a difference in perceived and actual value. Now it’s time to collect some evidence in the form of data. Each student plays 10 games on a 60 second timer. Take an average of the class scores. Now increase the time to 90 seconds. Students play another 10 games each. What is the class average now? How much is each extra second worth? This is also a great opportunity to discuss whether or not there might be any bias in the data.

- **Would you like to buy another number?**

Some target numbers are simply not possible with 6 numbers. An endless supply of numbers would make every target number achievable and simple. The journey from 6 numbers to an endless supply is one of increased simplicity, but how much is that journey worth in regards to points? Students can conduct a similar class experiment to the 'more time' option. Students will also need to consider restrictions on small and large numbers, do they have different point values?

- **Would you like to buy a mathematical operation?**

This represents a great opportunity to introduce students to a range of mathematical operations including:

$!$, 2 , $\sqrt{\quad}$, nC_r , nP_r , other ... the list is almost endless. Consider other commands on the calculator such as Mod(), Int(), GCD(). Students can experiment with things like: $5! \div 3!$, what a great way to get 20, or $6! \div 3!$ or use combinatorics and show students Pascal's triangle.

Students can explore one or more of the above, or create their own additional rule for the game, such as 'stop', an option to exit early and trade in the extra time as points. Stop may be called whether the target score has been reached or not.

Example:

Target Score: 826. Large: 25, 50. Small: 3, 5, 9, 10

Approximate solutions: $(50 + 25 + 9) \times 10 - (3 \times 5) = 825$ or $(75 + 5) \times 10 + 3 \times 9 = 827$

Suppose these approximate solutions took only 30 seconds to create, calling stop would therefore result in a score of $100 - 1 + 30 = 129$. [-1 = difference in the solution and target, +30 for time left on the clock.]

All the explorations involve gathering extra data to validate their recommendations for point scoring. More games equate to more practice, which returns us to students practicing their basic number skills.

For students that have never seen, nor heard of the Television show, some episodes are available on YouTube:

<https://youtu.be/SNk0OgMMxAq>

More details about the original game show can be found at:

https://en.wikipedia.org/wiki/Letters_and_Numbers

Some of the extension opportunities in this activity include modifications to the program. The 10 minutes of coding section of the Texas Instruments Australia website contains a series of free lessons that will help students extend their program.

<https://education.ti.com/en-au/activities/ti-codes-overview>

There are also numerous videos on the Texas Instruments Australia YouTube channel. A separate playlist for STEM includes lots of coding tutorials:

<https://bit.ly/TI-Codes>

Introduction

Letters and Numbers was a very popular television show on SBS. The game show consisted of two main games. The word game required participants to build the longest word from nine randomly generated letters. The letters consisted of a nominated combination of vowels and consonants. The number game consisted of a total of six numbers, some small and some large. The numbers must be put into an equation to form a result as close as possible to the three-digit target number.

In this activity you will write a program that will simulate the "numbers" component of this popular television show.

Creating a Program

Start a new TI-Nspire document and insert a New Program.

Call the program: Numbers

The large numbers in the game consist of: {25, 50, 75, 100}.

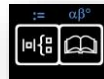
Small numbers consisted of: {1, 2, 3, ... 10}

These sets of numbers can be stored:

Large:={25,50,75,100}

Small:={1,2,3,4,5,6,7,8,9,10}

The “:=” symbol is located above the maths templates.



The next step for the program is to request the quantity of small numbers, a value between 3 and 5.

menu > I/O > Request

The request command consists of a combination of text and variable.

Request “text prompt”, variable

For this request, a prompt such as “How many smalls” is sufficient. Store the result in ‘n’. (variable)

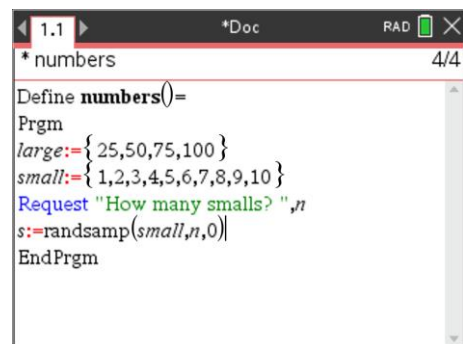
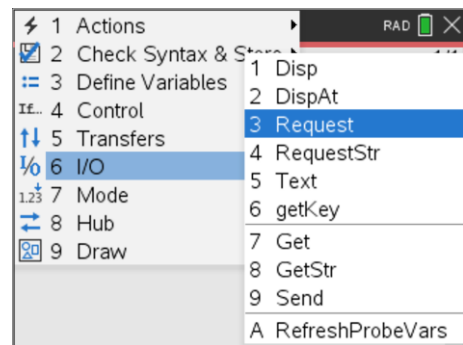
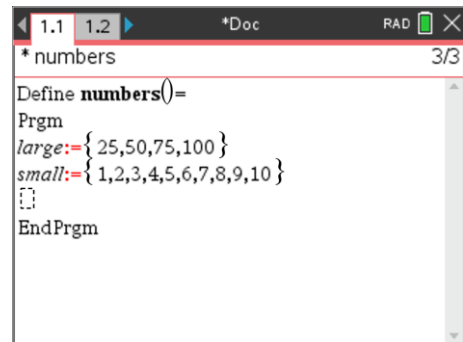
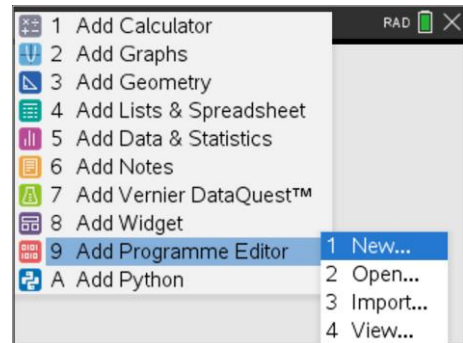
The random sample command has the following syntax:

RandSamp(list,quantity,repeats)

The sample is taken from the small numbers, the quantity of numbers to be sampled is ‘n’. If repeats = 1, digits will not be repeated, however if repeats = 0, then repeats may occur.

The RandSamp command can be accessed from the catalogue. The sample is stored in variable ‘S’.

Repeat this instruction using the appropriate list, quantity of numbers and corresponding repetition for large numbers. Store the result in “L”.



The program is now ready to display some values to the user.

The first item to be displayed is the target value. The target value lies between 100 and 999.

[menu] > I/O > Disp

The random integer (whole number) is available from the catalogue.

Disp "Target Number ", RandInt(101,999)

Include similar display statements to show the selection of large (L) and small (S) numbers.

The final step is to include a timer! The "wait" command can be used as a timer.

[menu] > Control > Wait

The television show provided just 30 seconds, in this game you have 60 seconds. Set the wait timer to 60.

Program execution halts while the wait command is active. Immediately after the wait command, insert another display command to instruct the user to "stop".

Once you have finished writing your program, press **Ctrl + R** and start playing.

```

1.1 | *Doc | RAD | 7/7
---|---|---|---
* numbers
Define numbers()=
Prgm
large:={25,50,75,100}
small:={1,2,3,4,5,6,7,8,9,10}
Request "How many smalls? ",n
s:=randsamp(small,n,0)
l:=randsamp(large,6-n,1)
Disp "Target Number: ",randint(101,999)
{}
EndPrgm

```

```

1 Actions
2 Check Syntax {
3 Define Variable:
4 Control
5 Transfers
6 I/O
7 Mode
8 Hub
9 Draw
Disp "Small Numbers:
wait 60
{}
D Wait

```

Question: 1.

Test your program to make sure it works.

- a) Write down the target number and list of numbers generated.

Answer: Answers will vary, they are 'random'. Indeed, this is a great way to check that students are submitting their 'own' work.

- b) Write down your best attempt at a solution.

Answer: Answers will vary, again depending on the selection, however this is the part where you get to see whether students are following the correct order of operations. Students can be asked to submit their 'scratchings' (just as important). Watch the YouTube video to see how Lilly Serna formulated her answers, then consider how they should be written formally with appropriate notation.

In this version of the game you get a score. 100 Points is obtained if you get the score exactly. For every unit you are away from the target value, you lose a point.

- c) Write down your score.

Answer: Answers will vary, but should correspond to the difference between the student answer and the target value.

Question: 2.

Repeat Question 1 four more times, including the calculation of your score, then calculate your average score.

Answer: The purpose of this question is to set the structure in the event that this activity is being used as part of a bigger investigation, as outline in the teacher notes on page 1.

Question: 3.

For any game that you played with a score less than 100 points, determine if it was actually possible to get a perfect score, had more time been available.

Note: This may take some time!

Answer: It is not be possible to generate a solution for every problem. Consider the unlikely game: Target = 999. Large = 25, 50, Small = 1, 1, 1, 1. Of course this is an extreme case, but there will be many that are not possible. In the actual game show, it is not likely that a small number would have repeated more than twice. A similar problem: Target = 999. Large = 25, 50, Small = 1, 2, 3, 4 is possible. $(25 - 2 - 3) \times 50 - 1 = 999$. Obtained without even using the 4.

Extension**Question: 4.**

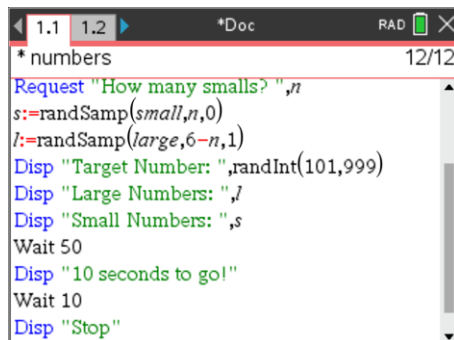
Are all target numbers achievable when 2 large and 4 small numbers are used?

Answer: This is essentially impossible for students to “prove”, however the intention is for students to spend more time on any one problem to exhaustively search for a solution. Some problems could be set as weekly challenges.

Question: 5.

Edit your program to warn the user when there is only 10 seconds left on the timer.

Answer: This is a relatively straight forward modification to the program.



```

1.1 1.2 *Doc RAD X
* numbers 12/12
Request "How many smalls? ",n
s:=randSamp(small,n,0)
l:=randSamp(large,6-n,1)
Disp "Target Number: ",randInt(101,999)
Disp "Large Numbers: ",l
Disp "Small Numbers: ",s
Wait 50
Disp "10 seconds to go!"
Wait 10
Disp "Stop"

```

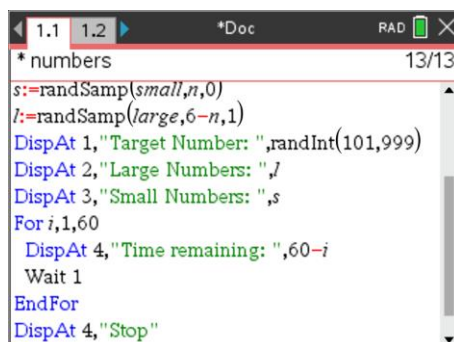
Question: 6.

Edit your program to provide a countdown timer for duration of the game.

Hint: The “display at” (dispat) command repeatedly displays the output on the same line.

Example: Dispat 2,value (where value is the amount of time remaining)

Answer: Changing Disp to DispAt and adding a loop makes for a great countdown timer.



```

1.1 1.2 *Doc RAD X
* numbers 13/13
s:=randSamp(small,n,0)
l:=randSamp(large,6-n,1)
DispAt 1,"Target Number: ",randInt(101,999)
DispAt 2,"Large Numbers: ",l
DispAt 3,"Small Numbers: ",s
For i,1,60
  DispAt 4,"Time remaining: ",60-i
  Wait 1
EndFor
DispAt 4,"Stop"

```

Question: 7.

Edit your program so the user can enter the amount of time they want to determine a solution.

Answer: A simple request statement for the time is not all that is required, students must also edit aspect of the timer within the program. The sample below shows the changes to the countdown time also.

```
* numbers 11/14
Request "How much time? ",t
s:=randSamp(small,n,0)
l:=randSamp(large,6-n,1)
DispAt 1,"Target Number: ",randInt(101,999)
DispAt 2,"Large Numbers: ",l
DispAt 3,"Small Numbers: ",s
For i,1,t
  DispAt 4,"Time remaining: ",t-i
  Wait 1
EndFor
```

Question: 8.

Edit your program to ensure the user only selects between 3 and 5 small numbers.

Answer: A while loop can force the request to prompt repeatedly while the value of n is not within the desired range.

```
"numbers" stored successfully
Define numbers()=
Prgm
large:={ 25,50,75,100 }
small:={ 1,2,3,4,5,6,7,8,9,10 }
n:=0
While n<3 or n>5
Request "How many smalls? ",n
EndWhile
Request "How much time? ",t
s:=randSamp(small,n,0)
```

Question: 9.

Edit your program to allow the user to select the quantity of numbers they have in order to solve the problem.

Answer: In the program below, users can select any quantity of numbers, but in the original game show, large numbers were not repeated, so the While loop ensures that no more than 4 large numbers exist in the result. This is particularly important as the program doesn't permit repeats in the sample command, so allowing the quantity of large numbers to exceed 4 would also generate an error.

```
"numbers" stored successfully
Define numbers()=
Prgm
large:={ 25,50,75,100 }
small:={ 1,2,3,4,5,6,7,8,9,10 }
n:=0
Request "How many numbers",q
While q-n>4
Request "How many smalls? ",n
EndWhile
Request "How much time? ",t
```