Senior Coding for Good, Badge 1 requirements:

Self-portrait
1. Use functions to create a self-portrait.
2. Write code to create a portrait.

Quiz
1. Learn about computer logic.
2. Explore **"If"** statements.
3. Use computer logic to create a quiz show.

Be sure to review the "Senior Coding for Good" overview/guide for the badge requirements and badge steps provided by your leader and the Girl Scouts. It also includes relevant vocabulary and interesting background information to spark your interest in coding. This lesson will allow you to earn **Badge 1: Coding Basics** using your TI-Nspire™ CX II graphing calculator.

Introduction

The programming language **BASIC** (stands for **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode) was developed in the 1960s as an easy system for teaching computer programming. TI-Basic is similar to other flavors of BASIC, but you must select the programming words and commands from the onboard menus, as you will soon see.

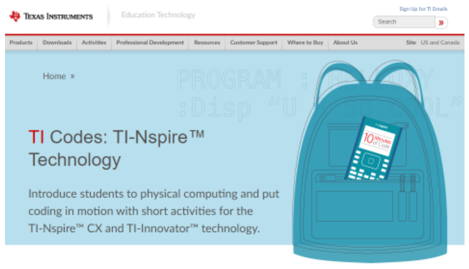
The Senior Level, Badge 1: Coding Basics consists of two separate programming projects:

- Part 1: A graphics program to produce a self-portrait, and
- Part 2: A quiz program to help someone learn or review facts or a skill

Note: The DRAW features used in this self-portrait project require a TI-Nspire™ CX II graphing calculator.

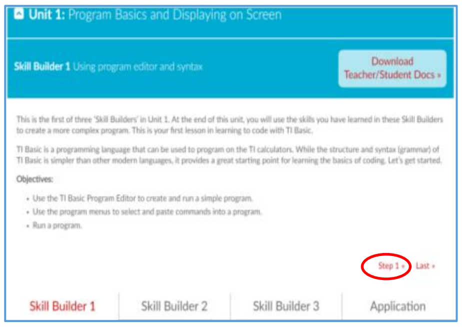
1. For an introduction to programming on the TI-Nspire™ CX II graphing calculator, see the TI Codes lessons at **education.ti.com > Activities > TI Codes > TI Basic**. Units 1 through 4, and Unit 6 (Drawing), should be enough to get started. Then return to this lesson for the **Senior Coding, Badge 1** project. Go to TI Codes (for TI-Nspire™ technology).

   How to navigate TI Codes: Be sure to notice that each unit has three Skill Builders (SB) and one Application activity, as you can see here. You will navigate through each Skill Builder by clicking on the "Step 1" in the right bottom corner, which will take you through the content. This screenshot shows you are currently in Skill Builder 1 > Step 1. After you complete all the steps in SB 1, you will then move to SB 2, and so on, until you complete all of Unit 1. You will then move on to Units 2, 3, 4 … and do the same thing.



TI Codes: TI-Nspire™ Technology

Introduce students to physical computing and put coding in motion with short activities for the TI-Nspire™ CX and TI-Innovator™ technology.



Unit 1: Program Basics and Displaying on Screen

Skill Builder 1 Using program editor and syntax

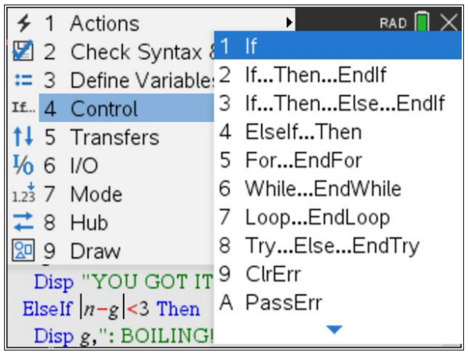
This is the first of three 'Skill Builders' in Unit 1. At the end of this unit, you will use the skills you have learned in these Skill Builders to create a more complex program. This is your first lesson in learning to code with TI Basic.

TI Basic is a programming language that can be used to program on the TI calculators. While the structure and syntax (grammar) of TI Basic is simpler than other modern languages, it provides a great starting point for learning the basics of coding. Let's get started.

Objectives:

- Use the TI Basic Program Editor to create and run a simple program.
- Use the program menus to select and paste commands into a program.
- Run a program.

- After you have completed the TI Codes units you know about:
- The programming process (planning, coding, testing and debugging)
- Using the TI-Nspire™ Basic Program Editor and its menus
- Running a program
- Some important pieces of TI-Basic code including: Request, Disp, variables and assignment statements, **If…Then…Else** structures and loops, like **For** and **While** (some of these statements found on the [menu] key are shown to the right)
- **Draw** features in the programming world of **TI-Nspire™ CX II**



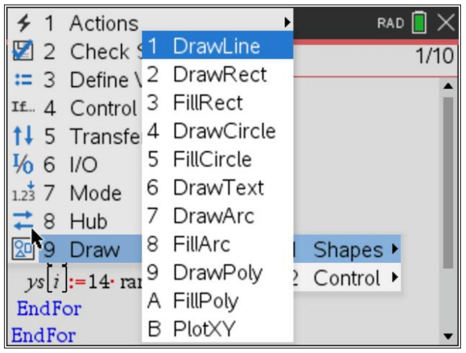
```
Disp "YOU GOT IT
ElseIf |n-g|<3 Then
Disp g,": BOILING!
```

2. **Part 1: Self-portrait.** Plan your project on paper first!

There are several **functions** on the [**menu**] > **Draw** > **Shapes** menu shown to the right that let you design interesting graphic images.

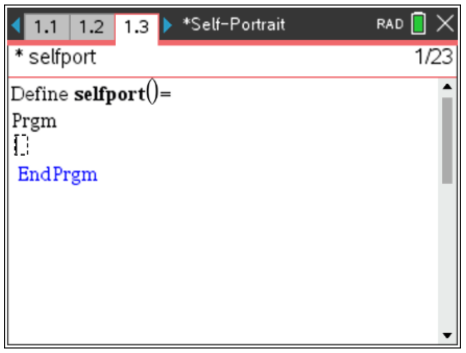
These tools are covered in Unit 6: Drawing of the TI Codes lessons you studied.

These functions produce points, lines, rectangles, circles and ellipses, polygons and text in the colors of your choosing.



```
ys[i]:=14· ra
EndFor
EndFor
```

a. Begin with a new document (**[home] > New**), and Add Program Editor. Name the program **selfport**.



```
* selfport                         1/23
Define selfport()=
Prgm

EndPrgm
```

b. To refresh these skills, use the statement

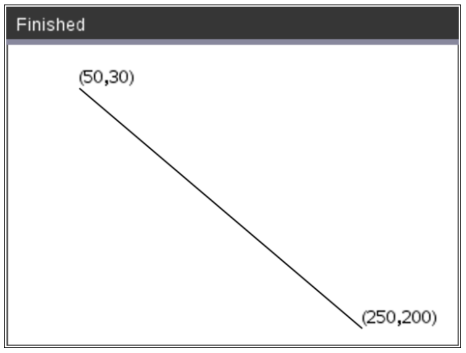
DrawLine x1, y1, x2, y2

to draw the segment between the points (x1, y1) and (x2, y2)

Example: DrawLine 50, 30, 250, 200

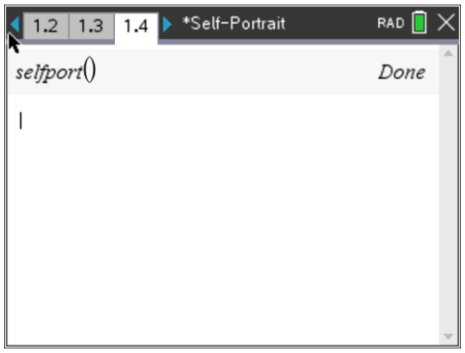
Run the program by pressing [**ctrl**] -**R** and then [**enter**] and you see a diagonal segment on the screen.

Note: To also display the two pairs of coordinates as shown here, that is another command. They are shown here for location purposes only.



c. To return to the Program Editor, press any key to remove the graphics canvas and return to the Calculator app (shown).

Either press **[ctrl]-[leftarrow]** or use the mouse to click on the previous page tab at the top of the screen (page 1.3 in the image).



```
selfport()                Done
```

d. To draw a red rectangle:

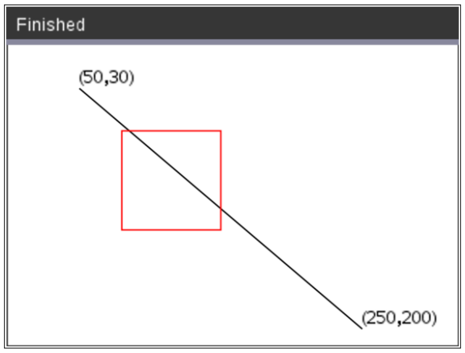
SetColor r, g, b

(r, g and b must be integers between 0 and 255)

DrawRect *x, y, width, height*

(x, y) is the location of the upper left corner

Example: SetColor 255, 0, 0
DrawRect 80, 60, 70, 70



Run the program by pressing **[ctrl] -R** and then **[enter]**. It Makes the red *square* in this image since the width and height are both equal to 70.

e. Let's try this. To make a solid (filled) circle,

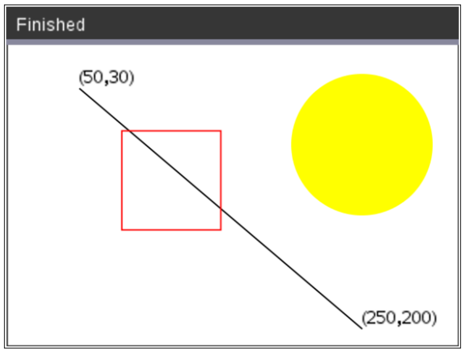
FillCircle x-center,y-center, radius

Example to make a big yellow dot:

SetColor 255, 255, 0
FillCircle 250, 70, 50

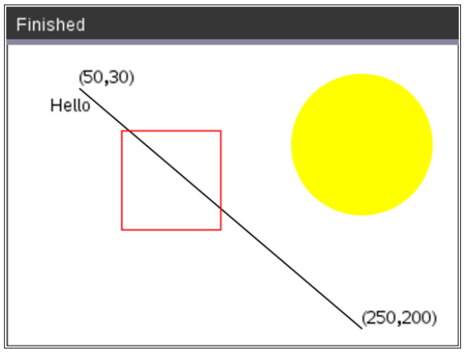
Note: The center is (250, 70) and the radius is 50.

Now run the program again. Does yours look like this?



f. Now let's try and add some text on the screen:

DrawText 30, 50, "Hello"



Note that (50,30) (the line's endpoint) and (30,50) ("Hello" position) are two *different* locations on the screen.

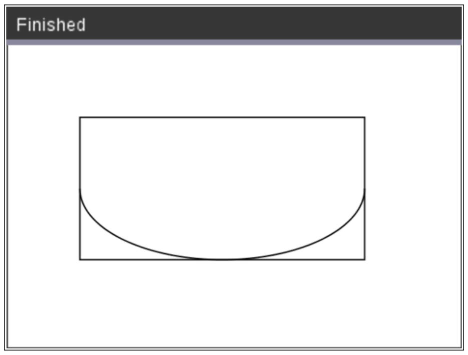
Draw the word "Hello" at the coordinates (30, 50). The coordinates are the lower-left corner of the text area. Is your "Hello" yellow? Use **SetColor** to change the color *before* the **DrawText** command.

This is how the coordinates were displayed earlier:

DrawText 50, 30, "(50,30)"

g. **DrawArc** and **FillArc** are a little trickier. (*We started a new program for this demonstration but you can continue with your program and add to your graphics.*) First make a rectangle (surrounding the arc), and you will get the idea:

```
DrawRect 50, 50, 200, 100
DrawArc   50, 50, 200, 100, 180, 180
```



Drawing an arc is like drawing a rectangle. The arc is *inscribed* in the rectangle, just touching each side. The last two values (180, 180) are the *StartAngle* and the *ArcAngle* to produce the lower portion of the ellipse shown. It starts at 180 degrees ("west") and goes another 180 degrees counterclockwise.

A complete circle or ellipse can have any *StartAngle* but should have an *ArcAngle* of 360 degrees. So, the **DrawArc/FillArc** commands need a "bounding rectangle" and an angle interval to make an arc. If the rectangle is a square then the arc will be part of a circle, otherwise it is part of an ellipse.

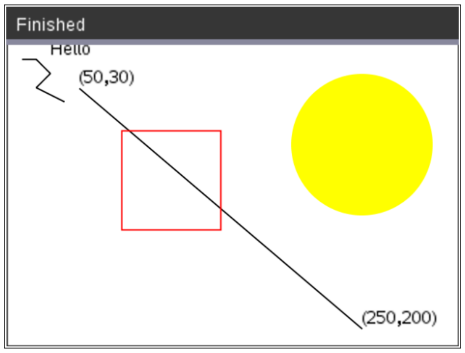
You can, of course, draw the arc without drawing the rectangle.
Try experimenting with other values for *StartAngle* and *ArcAngle*.

h. To draw a filled or outlined polygon (or sequence of segments, a "poly-line") create two lists, one for x-values and one for y-values:

```
xs:={10,20,30,20,40}
ys:={10,10,20,30,40}
```

then use the DrawPoly command:

```
DrawPoly xs,ys
```



Reminder: **:=** is found on **[ctrl]-[math templates]** to the right of the 9 key.

Notice that there are *five* points which makes the *four* segments in the upper left corner of the screen.

Note: The word "Hello" appears in a different location in this image. Can you imagine how that happened?

i. A filled polygon is useful for drawing large portions of your self-portrait.

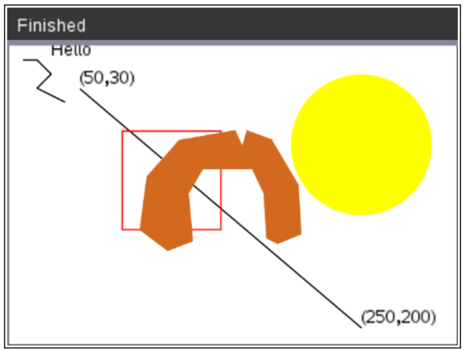
Example to draw some brown hair:
SetColor 210,105,30

Note: The following two lists will not fit on your screen. As you type the values, your screen will shift. Just keep typing and press **[enter]** *at the end.*

xs:={187, 206, 208, 191, 183, 181, 173, 138, 128, 131, 113, 93, 98, 121, 161, 166, 169, 188}

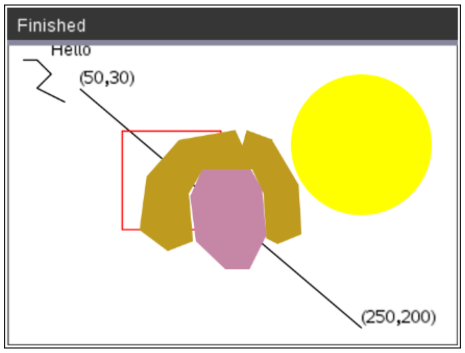
ys:={67, 99, 134, 141, 137, 105, 88, 88, 105, 139, 146, 131, 93, 67, 60, 71, 60, 67}

FillPoly xs,ys



Run your program. Does it look like this? You are learning very useful tools. You will soon be ready to draw your own "self-portrait."

j. Now, it's your turn! Write your own graphics program using a combination of Draw commands to create your own self-portrait program.



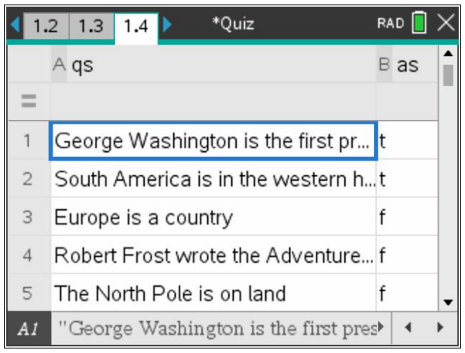
All you have to do is determine the correct x- and y-coordinates. Remember: **Plan first, code later.** Graph/grid paper will help.

Hint: the screen coordinates are: upper-left (0,0) → lower right: (317,211). On ordinary graph paper (1/4" grid lines) you might want to use a scale of 10 pixels per grid unit.

*Share your project on Instagram, or other social media, and be sure to tag it **@TICalculators***

3. **Part 2: Quiz**

A "Quiz program" presents the user with a series of questions to answer. In this sample project we use a true/false format. The program keeps track of the number of correct answers and reports a score at the end of the program. To prepare for our program we will store text "strings" in lists on the TI-Nspire™ CX II graphing calculator. Set up two lists, one containing questions and the other the answers.



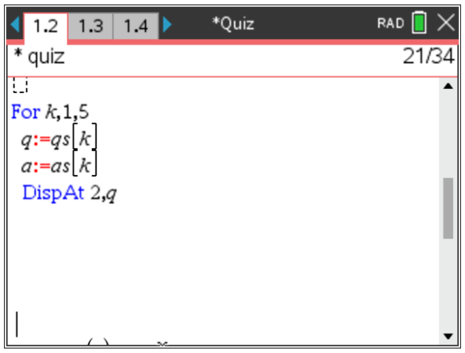
	A qs	B as
=		
1	George Washington is the first pr...	t
2	South America is in the western h...	t
3	Europe is a country	f
4	Robert Frost wrote the Adventure...	f
5	The North Pole is on land	f

A1 "George Washington is the first pres

In the screen to the right, in the **Lists & Spreadsheet** app, we made a list called **qs** (questions) and a list called **as** (answers: true or false). Each question and each answer is entered into the spreadsheet cells using "quotation marks." See the status line at the bottom of the image.

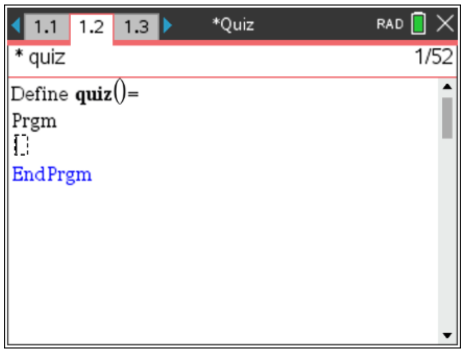
The list names (**qs** and **as**) are in the TOP cell above the lists and the formula cell (the second row on the screen) is left empty.

a. **The plan:**
Use a **For** loop to process each question **qs[k]**. Use **DispAt** statements to position each part of the question on the screen. Keep track of the number of correct answers. Report the score at the end of the quiz.



```
For k,1,5
 q:=qs[k]
 a:=as[k]
 DispAt 2,q
```

b. Start a new document (**[home] New**), and Add Program Editor from the new page menu. Name the program **quiz**.



```
Define quiz()=
Prgm

EndPrgm
```

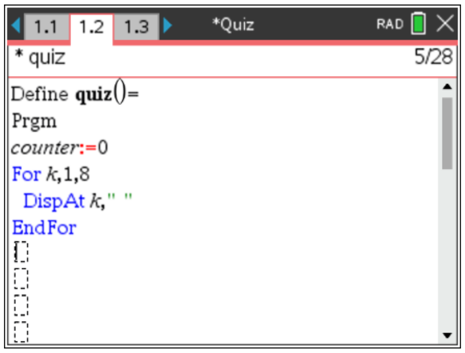
c. The variable **counter** is set to 0 to count the correct answers.

A preliminary **For** loop is used to clear the screen:

```
For k, 1, 8
    DispAt k, " "
EndFor
```

Hints: **[menu] > Control> For…EndFor**
[menu] > I/O > DispAt

Indentation is provided and recommended but optional.



```
Define quiz()=
Prgm
counter:=0
For k,1,8
 DispAt k," "
EndFor
```

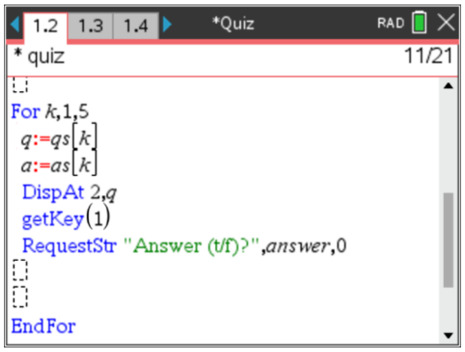
d. Now begin the main part of the program. After the clear screen loop from the previous step write another **For…EndFor** loop **([menu] Control**) that will:

i. Display the question (and answer choices if multiple-choice). The statement **q := qs[k]** stores the kth question in the variable **q.** Then the **DispAt** statement (**[menu] I/O**) displays the question on line 2.

ii. Pause the program using **getKey(1) ([menu] I/O)** *(user presses any key to continue).*

iii. Input the answer using **RequestStr** (**[menu] I/O**) *(user presses enter to answer).*

iv. See if the answer is correct and count the correct answers. *(This code is missing in the image to the right.)*



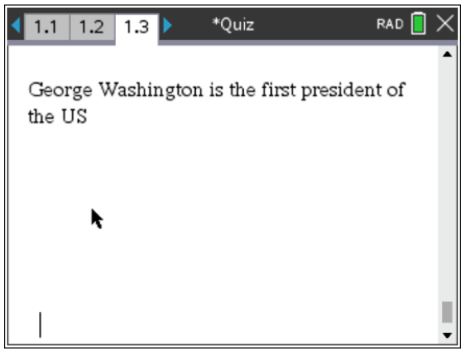
```
For k,1,5
 q:=qs[k]
 a:=as[k]
 DispAt 2,q
 getKey(1)
 RequestStr "Answer (t/f)?",answer,0


EndFor
```

Note: The first few steps are shown here, but this is not the complete program so, although the program runs, there is more to do:

- *Add an* **If** *statement in the loop to see if the answer is correct and count the number of correct answers*
- *Add statements below the loop to report the final score*

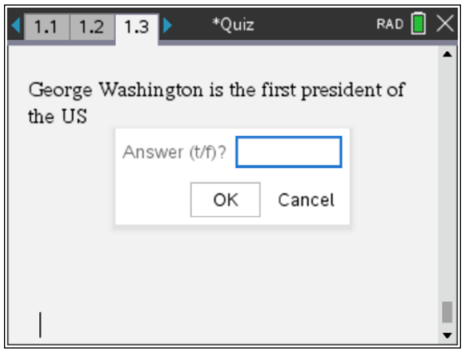
e. The statement **getKey(1)** pauses the program until a key is pressed *(see screen to the right)* to give the scout ample time to study the question before answering. **getKey()** is on **[menu] I/O**.

Note: See the challenges at the end of this document.



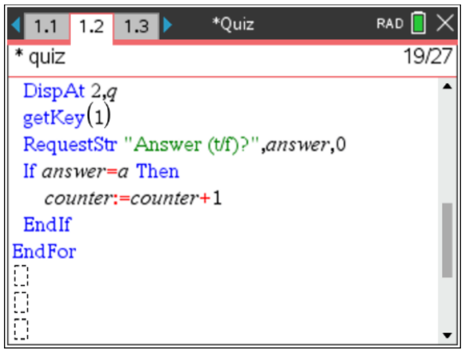
```
George Washington is the first president of
the US
```

f. The **RequestStr** statement gets a response from the user. This produces a dialog box in the center of the screen for inputting an answer.



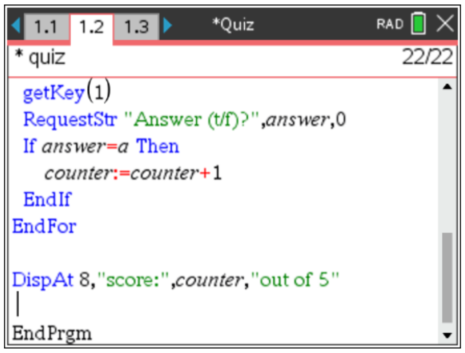
```
George Washington is the first president of
the US
Answer (t/f)?
OK   Cancel
```

g. Use an **If** statement to determine if the user's answer is right, and update a counter accordingly.



```
* quiz                                19/27
 DispAt 2,q
 getKey(1)
 RequestStr "Answer (t/f)?",answer,0
 If answer=a Then
   counter:=counter+1
 EndIf
EndFor
```

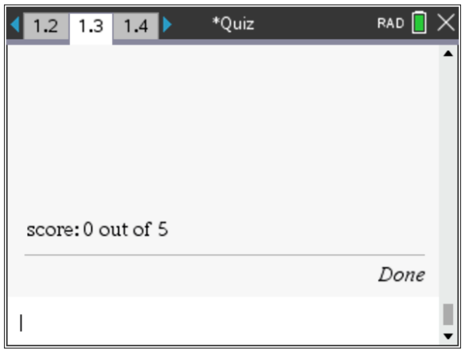
h. After the **End** of the **For** loop (the end of the quiz), report the results.



```
* quiz                                22/22
 getKey(1)
 RequestStr "Answer (t/f)?",answer,0
 If answer=a Then
   counter:=counter+1
 EndIf
EndFor

DispAt 8,"score:",counter,"out of 5"

EndPrgm
```

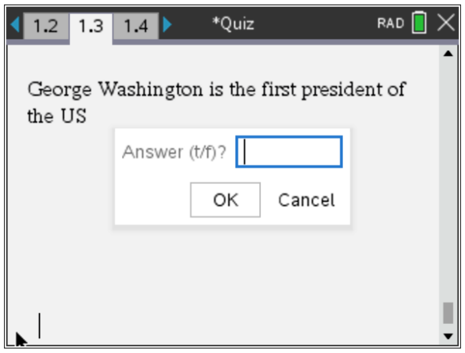
i. When you run the program (press **ctrl-R**) each question is displayed one at a time. Press a key to see the input box, type an answer and **[enter]**.

At the end of the quiz the score is displayed.



```
score: 0 out of 5
                                   Done
```

j. Using this program as a sample, you can improve the project in many ways. Get creative! How about a quiz about the Girl Scouts? 😉


George Washington is the first president of the US

After your program is finished, and before sharing the file with others, you can remove the **Lists & Spreadsheet** app to hide the questions and answers from the user. On the **Lists & Spreadsheet** app, press **ctrl-K** to select the app, then press **[del]** to remove the app from the document. The lists will *remain* in the document.

In the future, you can always insert a new **Lists & Spreadsheet** app again, and add the stored lists to the spreadsheet to edit the questions and answers.

*Share your project on Instagram, or other social media, and be sure to tag it **@TICalculators***

Congratulations!

You have completed the requirements for earning your **Senior Coding for Good, Badge 1: Coding Basics**. Now that you have completed this requirement, you are challenged to *give service* by *sharing* what you have learned about coding with others. Refer to the "Coding for Good" Girl Scout guide for suggestions on how to do so.

What's next? (Optional extensions)

Ready to try some additional practices with your new skills?

1. In the Quiz project:
   a. Create a multiple-choice quiz with three to five choices for each question. The correct answers will be the letters **a**, **b**, **c** ….
   b. To set up a multiple-choice quiz using three-to-five-answer choice lists such as, **bs**, **cs**, **ds**, **es**, then, in your program ask the user to enter the letter of their choice.
   c. Offer multiple tries ("Wrong, try again."); give full credit for getting it right on the first try and give partial credit on later tries. This will take an extra loop in the response section of the program.
   d. Display a percent score at the end.
   e. Display the questions in a random order.