Ready Set…Solve!

Ready, set….solve! You will create a fast-paced game to help practice solving multi-step equations. If the player's math skills are on point, they will out run the computer. Make too many mistakes, and the calculator will win the race. Your code will generate questions for the user to answer until either the user or the computer wins. Once there is a winner, the computer will display the race and declare a winner.

Objectives:

Programming Objectives:

- Use functions to generate strings and integers
- Use the randint() function to generate random integers.
- Use the TI-PlotLibrary library to plot points
- Use the sleep command from the Time library to animate the plot
- Use loops to repeat code
- Use if statements to make selections

Math Objectives:

- Practice solving multi-step equations.

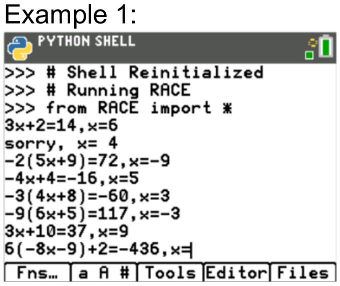
Math Course Connections: This activity is recommended for Pre-Algebra or Algebra 1.

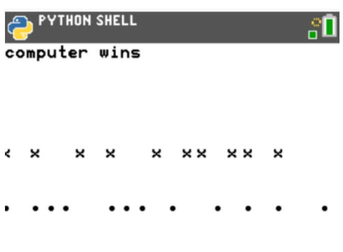
In this program, you will generate random equations in three different forms.

Form 1: Two-step equations such as: $3x + 8 = 14$ or $5x – 1 = -16$
Form 2: Distribution equations such as $3(2x – 5) = -9$
Form 3: Distribution equations with an additional coefficient such as $-2(4x – 1) + 2 = 4$

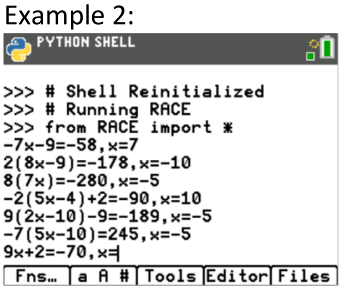
The program will ask a series of questions. If the user answers a question correctly, the user will "run" fast. If the user answers a question incorrect, they slow down. The computer answer runs each leg of the race using a random number generator.

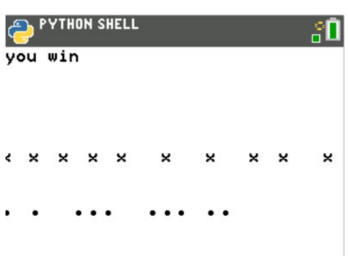
Example 1:





Example 2:

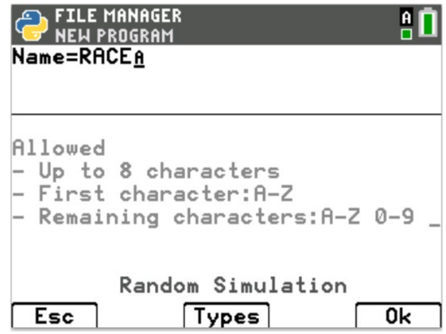




1. The first step will be to create a python Random Simulation document.

   Create a new python project named “race”.

   Select “Random Simulation” from the type menu.



2. You will need two more libraries, the TI Plot Library and Time library.

   Place your cursor on the line below the **from random import I**
   Fns > Modul > ti_plotlib > import ti_plotlib as plt

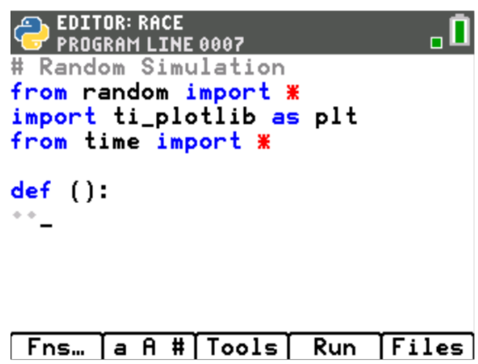
   Fns > Modul > Time > from time import *

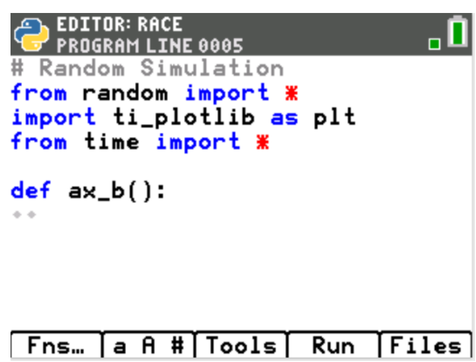
3. The race game will generate various single variable equations to solve. The most basic form, will be of the form ax + b = c.

   To help minimize code later, you will create a function definition.

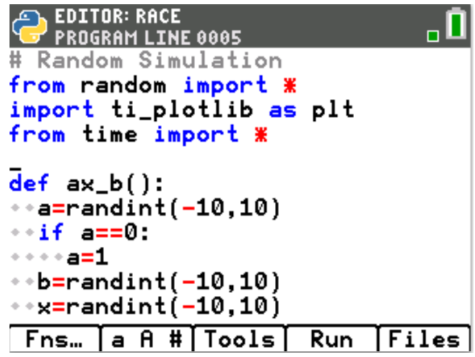
   Fns > def function

   Change the name of the function to ax_b. Leave the argument empty. The “_” key is in the [a A #].





4. This function generates equations in the form ax + b = c.
   You will generate random integer values for a, x, and b.
   You will generate integers between -10 and 10. If a is 0, set a = 1.



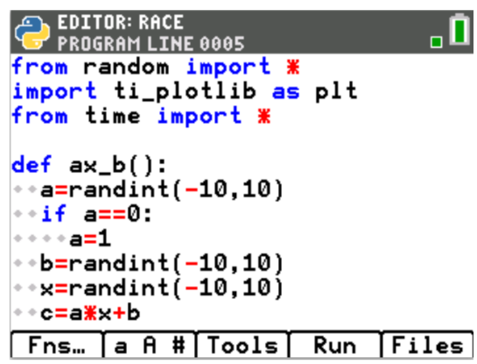

```
a = randint(-10,10)
if a == 0:
  a = 1
b = randint(-10,10)
```

Fns > Modul > random > randint
Fns > Ctl > if

Notice the diamond pattern in the example on the right. All lines that are in definition have indentation depths of at least two diamonds. This tells the computer the lines are part of the definition. The line a = 1, needs to have 4 diamonds because this line is part of the definition AND it is part of the if statement.

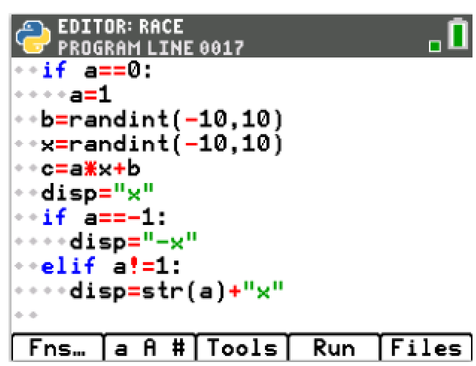
**Hint: The tools menu, [zoom], contains both a copy line and a paste line command. This could be helpful when creating variables a, b and x.

5. The value of c is ax + b. Add the line
   c = a*x + b



In math, you can write c = ax + b. However, in programming, this will cause an error. The compiler thinks ax is a variable. You must type c = a*x+b so the compiler knows to multiply the variable a with the variable x.

6. Now to create a string display to show the equation.



If a is a 1, you should display just “x”.
If a is a -1, you should display “-x”.
For any other values of a, you should display “ax”.

We can assume the display is “x”, then over write the value if necessary.

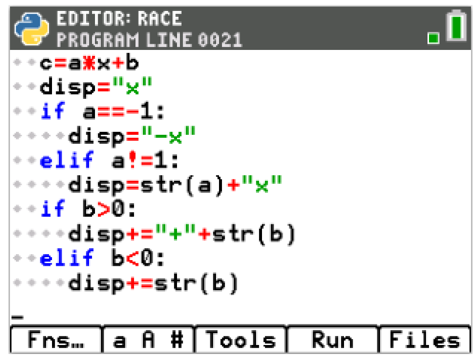
```
disp = “x”
if a == -1:
```

```
        disp = "-x"
    elif a != 1:
        disp = str(a) + "x"
```

Since a is an integer and "x" is a string, you must change the integer to a string before adding the two together using str(a) + "x".

Fns > Ctl > if
Fns > Ctl > elif
Fns > Type > str
[2nd] [math] has both the == and the !=

7. If b is not a 0, add b to the display.

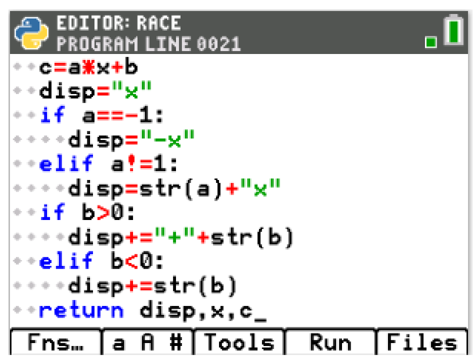


If b is positive, you need to add both a "+" and the string value of b.
If b is negative, you only need to add the string value of b, this will also add the needed "-" sign by default.

```
if b > 0:
    disp += "+" + str(b)
elif b < 0:
    disp += str(b)
```

*Make sure you used the += after display. You want to add to the existing display not replace it.

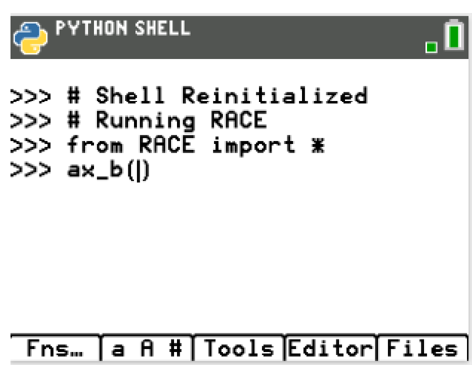
8. Lastly, return the disp, x, and c.




```
return disp, x, c
```

Fns > return

9. Let's try your function out.

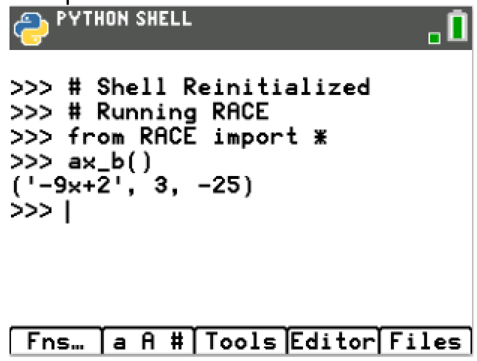


Execute your program [Trace]

The screen will look blank. Press the [var] key and select ax_b
Or
Type ax_b().

Press [enter]

Sample:



The sample to the right displays:
-9x+2, 3, -25

That means
-9x+2=-25…….the value for x should be 3. Let's verify by substitution.

-9*3 + 2 =
-27 +2 =
-25 = -25

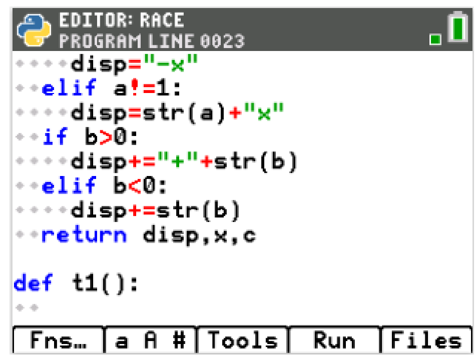
Make sure your values work as well.
We successfully have a function to generate ax+b, x, c

10. Now that you have a function to generate ax+b, you can use this function to generate equations of the form ax+b=c and d(ax+b) = c.

    Let's write the first type.

    Create a new function named t1

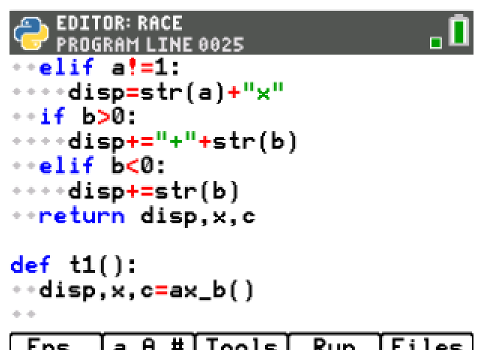
    Fns > def function



11. Use the ax_b function to generate the left side of the equation, the x and c.

    disp, x, c = ax_b()

    **Make sure you have commas between disp, x and c.
    Did you place a () behind ax_b?

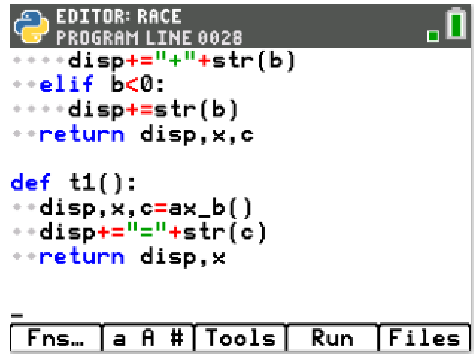


12. You need to add the "=c" to the end of the display. c is an integer, so you will need to change it to a string before you can add it to "=".

    disp += "=" + str(c)

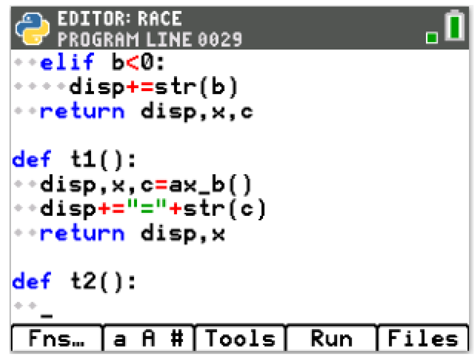
13. Now, return both the display and the x value.

    return disp, x



14. Notice your t1 function only has three lines of code. The function ax_b() did a lot of the work. Your t2 function will be of the form d(ax+b) = c. It will also use the ax_b information to make coding shorter.

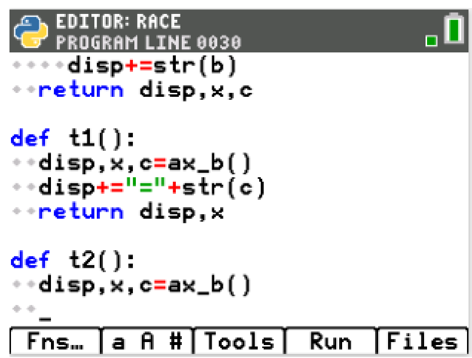
    Add a function named t2.



15. Use the ax_b function to generate the left side of the equation, the x and c.

    disp, x, c = ax_b()

    **Make sure you have commas between disp, x and c.
    Did you place a () behind ax_b?

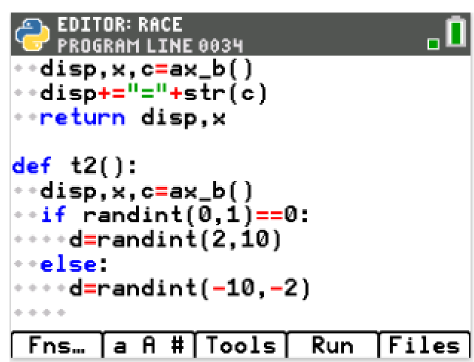


16. This equation type will be of the form d(ax + b) = c.
    We want d to be between [2,10] or [-10,-2]

    You will use an if..else statement to make this happen.
    You will "flip a coin", if randint(0,1) == 0, d will be positive, otherwise d will be negative.

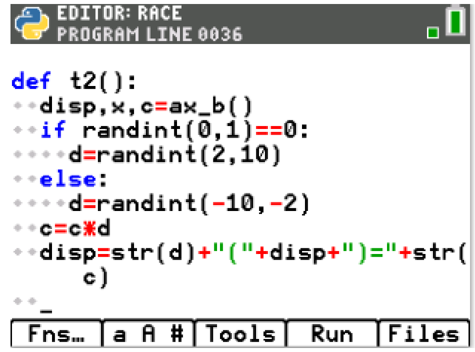
```
if randint(0,1) == 0:
    d = randint(2,10)
else:
    d = randint(-10,-2)
```

    *Remember, randint is Fns > Modul> random > randint



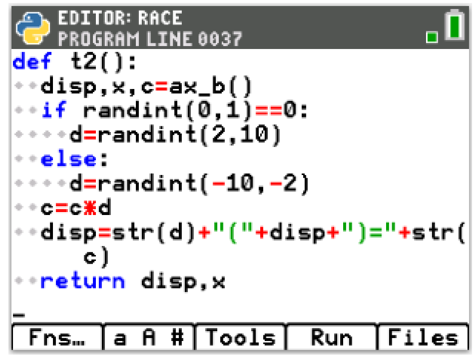
17. Multiply the total by d and add it to the display:

    c = c*d
    disp = str(d) + “(“ + disp + “)=” + str(c)



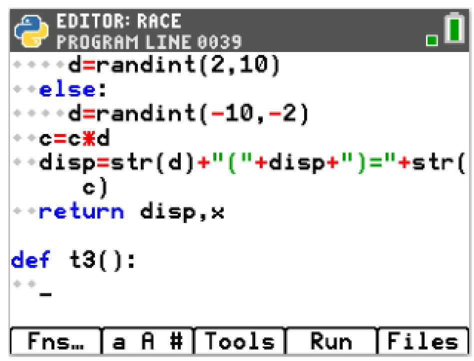
18. Return both the display and x.

    return disp, x



19. Using t2, you can easily make a t3 problem that looks like d(ax + b) + e = c.

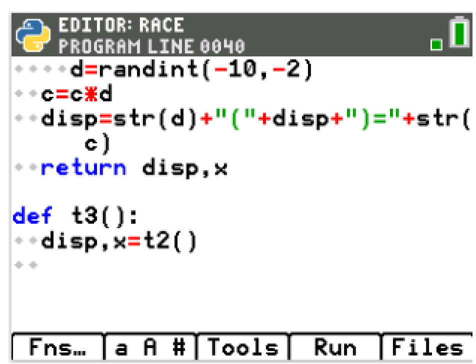
    Create a new function t3



20. The function t2 creates functions in the form d(ax+b)=c.
    It returns both the display equation and the value for x.
    You can add to this to make d(ax+b) + e = c.

    First, get the display and the value from x using t2.

    disp, x = t2()



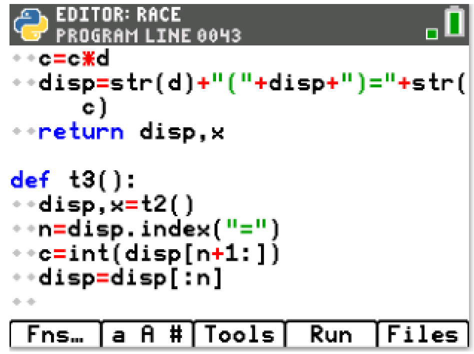
21. Currently display contains d(ax+b)=c.
    You need to break this apart into the left display and the value on the right. Then, you can generate a number, e, and add it to both sides.

    To break the string apart you will:
    - Find the location, n, of the "=" sign.
    - Store the numerical vale of the right side into a variable c.
    - Store the left side of the display up to the location of the "=" in disp

    You must find c using the original display value before you replace disp with only the left side of the display

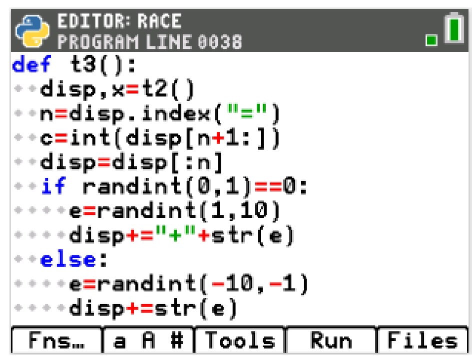
    Type:
    ```
    n = disp.index("=")
    c = int( disp[n+1:])
    disp = disp[ :n]
    ```

    [a A #] contains the symbols : and [ ].



22. e should be either [1,10] or [-10,-1]. Therefore the if statement is similar to the if for the variable d.

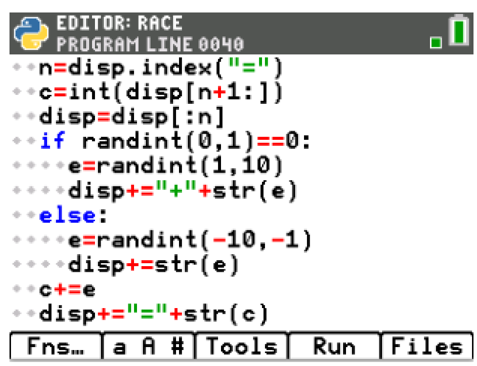
    ```
    if randint(0,1) == 0:
        e = randint(1,10)
       disp+="+"+str(e)
    else:
        e = randint(-10,-1)
        disp+=str(e)
    ```



23. Add the e to the total and the display.

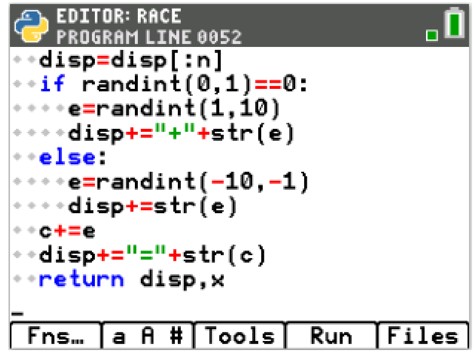
    ```
    c += e
    disp+="="+str(c)
    ```

    Make sure both of these lines are indented only one level, two diamonds. They are not part of the if statement. They are part of the definition.



24. Return both the new display and the value of x

    return disp, x



25. Now that you have functions to generate random questions, you are ready to code the race.

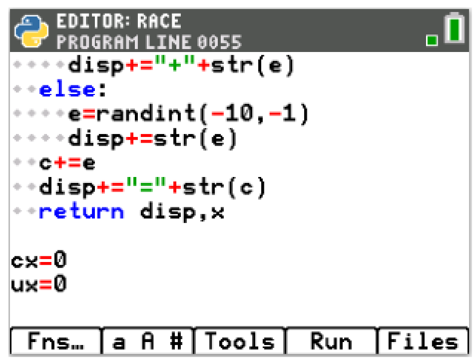
    The race layout will be:

    while both players have scores under 20:
        ask a question
        If the user is correct, add 2 or 3 steps, otherwise go 0 or 1 steps.
        The computer moves 1 to 3 steps.
    Display the race

26. Create distance run variables for the computer and the user.
    Set each one equal to 0.

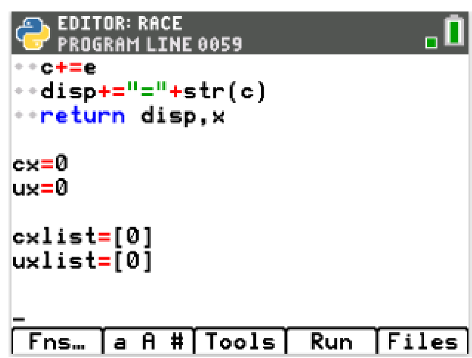
    cx = 0
    ux = 0

    *Hint: The tools menu, [zoom], has copy/paste line commands.



27. Add two list variables to keep track of each round.

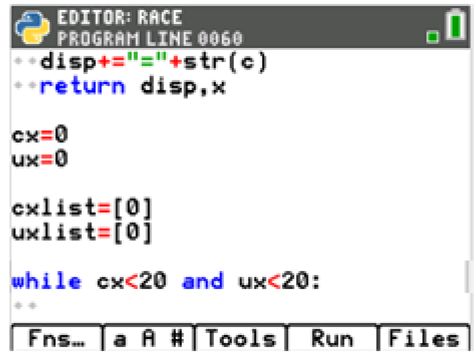
    cxlist = [0]
    uxlist = [0]



28. While both players have scores under 20, continue play.

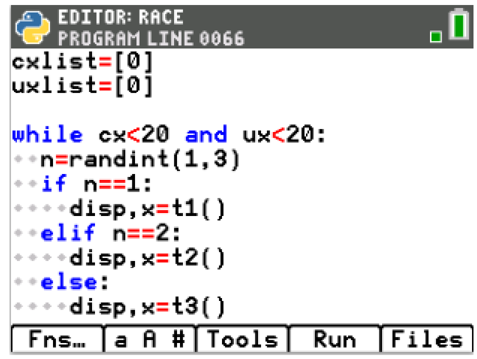
    while cx<20 and ux<20:

    Fns > Ctl > while



29. Generate a number n from 1 to 3. Use this number to use of the type functions to generate the question display and value for x.

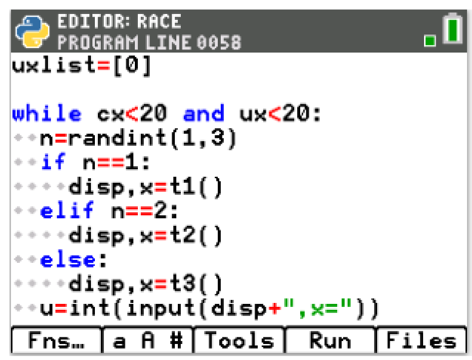
```
n = randint(1,3)
if n ==1:
    disp, x = t1()
elif n ==2:
    disp, x = t2()
else:
    disp, x = t3()
```



30. Ask the user the question. Store the response as u.

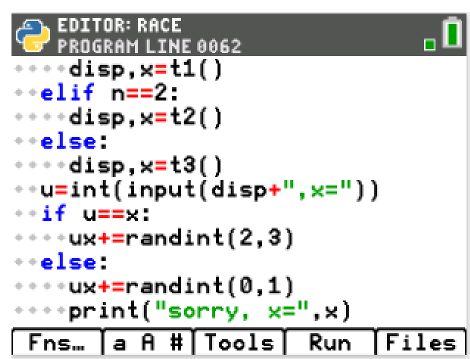
    u = int(input(disp+", x= "))

    Fns > Type > int
    Fns > I/O > input



31. If the user's u matches the x value, move the player forward 2 or 3 spaces. Otherwise, move forward 0 or 1 spaces.

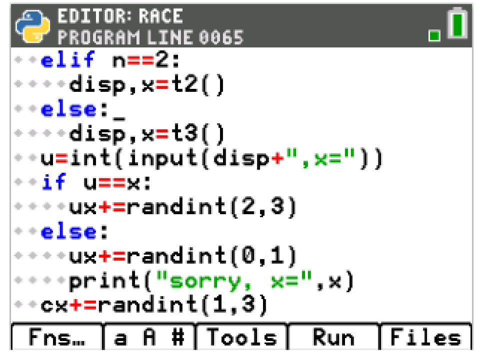
```
if u == x:
    ux += randint(2,3)
else:
    ux += randint(0,1)
    print("sorry x=",x)
```



**Check this carefully. Make sure you used u and ux correctly.

32. Add between 1 and 3 steps for the computer.

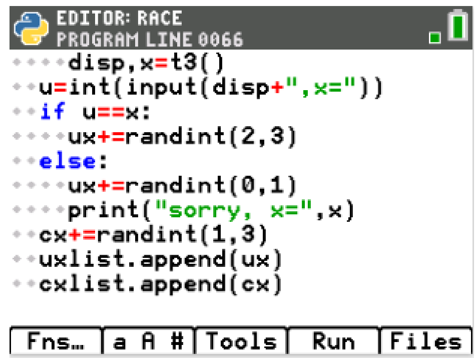
    cx += randint(1,3)



33. Add ux to uxlist and cx to cxlist.

    uxlist.append(ux)
    cxlist.append(cx)

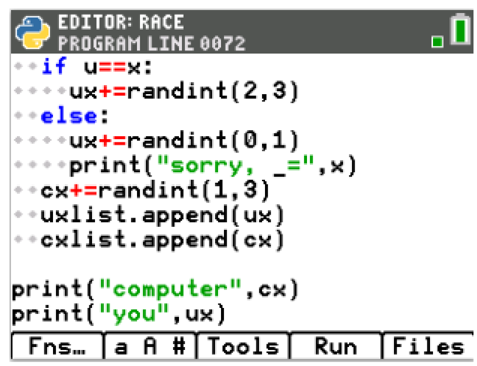
    Fns > Lists > append



34. After the loop, print the computer's score and your score.

    print("computer", cx)
    print("you", ux)

    Fns > I/O > print

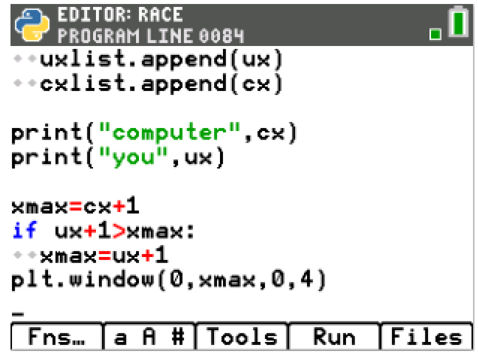


35. Execute your program. [trace]

    Who won?

    Run your program a few more times.

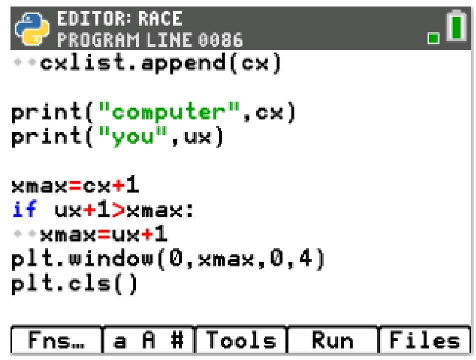
36. Now to add the “graphics”.
Add one to the length of the computer race. Set it equal to the maximum x. If the user ran farther, use that for the max x,




```
xmax = cx + 1
if ux > xmax:
   xmax = ux + 1
plt.window(0, xmax, 0, 4)
```

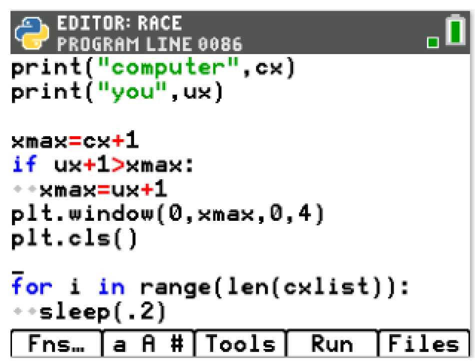
Fns > Modul > ti_plotlib > window

37. Clear the screen.



Fns > Modul > ti_plotlib > cls

38. You need to plot both lists. You could plot the lists all at once, but then there won’t be the suspense of watching the race. Therefore, you will plot one step of the race each 0.2 seconds.



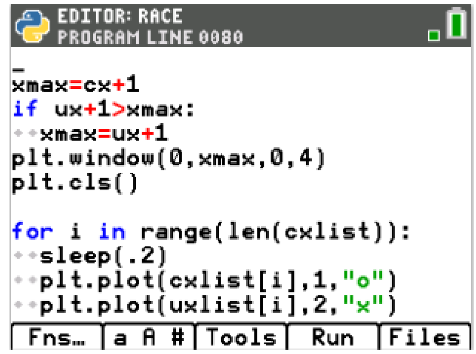
The number of items in the list can be found using len().

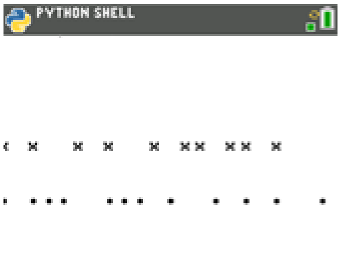
Add the line:

```
for i in range( len(cxlist) ):
    sleep(.2)
```

Fns > Control > for in range
Fns > > List > len

39. Plot each item in each list. plt.plot(xvalue, yvalue, mark)





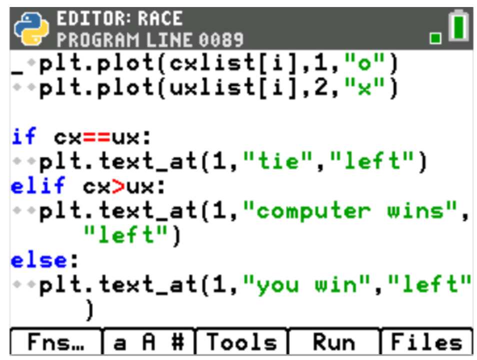
Let 'x' represent the user and plot the values at y = 2.

Let 'o' represent the computer and plot the values at y = 1.

plt.plot(cxlist[i], 1, "o")
plt.plot(uxlist[i], 2, "x")

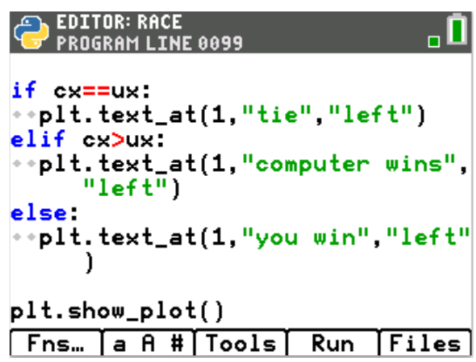
Fns > Modul > ti_plotlib > Draw > plot

40. Add an if statement that will print the winner.



if cx == ux:
    plt.text_at(1, "tie", "left")
elif cx > ux:
    plt.text_at(1, "computer wins", "left)
else:
    plt.text_at(1, "you win", "left)

41. Show the plot. Until the user presses [clear], the plot will stay on the screen.



plt.show_plot()

Fns > Modul > ti_plotlib > draw > show_plot

42. Play your game several times. How often can you beat the computer?

43. Challenge:
The command plt.color(red, green, blue) changes the color for both text and plots. The values for red, green and blue can be any integer value from 0-255.

example:
plt.color(255,0,0) and plt(100,0,0) are both red.
The 222 version will be a brighter red compared to the 100.

Use this function to change the “o” plots to blue or another color of your choice. Change the “x” plots to magenta or any other color of your choice. Change the who won text plots to match the x and o plots.