Solving Multi-Step Equations

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-Draw library to plot circles
- Use loops to repeat code
- Use if statements to make selections
- Use the sleep function to animate the plot

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 random probability.
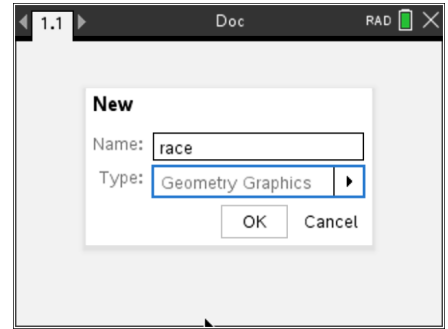
Example 1:

Example 2:

Math Explorations with Python

TI-NSPIRE™ CX II TECHNOLOGY

Ready Set…Solve!

1. The first step will be to create a python Geometry Graphics document.

   Create a new python project named "race".

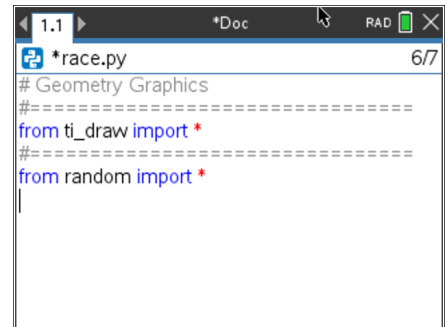   Select "Geometry Graphics" from the type menu.

```
1.1 ▶                Doc           RAD ☐ ✕

New
Name:  race
Type:  Geometry Graphics    ▶
                        OK    Cancel
```

2. You will need one more library, the random library.

   The random library contains the randint() function.

   Place your cursor on the line below the **import ti_draw**
   Menu> Random> from random import *

```
◀ 1.1 ▶              *Doc           RAD ☐ ✕
🐍 *race.py                            6/7
# Geometry Graphics
#============================
from ti_draw import *
#============================
from random import *
|
```
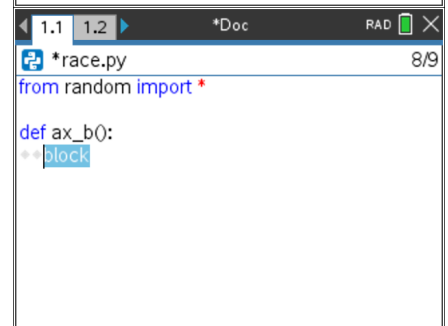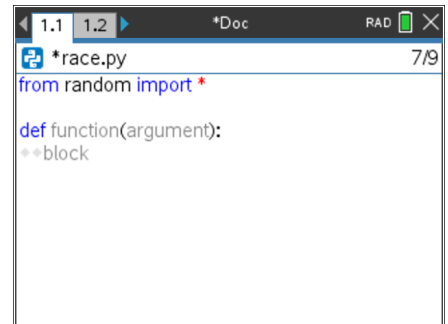
3. The game will generate various single variable equations to solve.

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

   Menu > Builts-ins > Functions < def function()

   Change the name of the function to ax_b. Leave the argument empty.
   The "_" key is in the ? key next to the G.

```
◀ 1.1  1.2 ▶          *Doc          RAD ☐ ✕
🐍 *race.py                            7/9
from random import *

def function(argument):
••block
```

```
◀ 1.1  1.2 ▶          *Doc          RAD ☐ ✕
🐍 *race.py                            8/9
from random import *

def ax_b():
••block
```

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)
    x = randint(-10,10)

    Menu > Random > randint
    Menu > Built-ins > Control > if

    Notice the diamond pattern in the example on the right. All lines that are in the 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.

```
◄ 1.1  1.2 ►        *Doc        RAD ▯ ✕
🐍 *race.py                          9/15
from random import *

def ax_b():
  ◆◆#ax+b=c
  ◆◆a=randint(−10,10)
  ◆◆if a==0:
  ◆◆◆◆a=1
  ◆◆b=randint(−10,10)
  ◆◆x=randint(−10,10)
```

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.

```
◄ 1.1  1.2 ►        *Doc        RAD ▯ ✕
🐍 *race.py                         14/16
from random import *

def ax_b():
  ◆◆#ax+b=c
  ◆◆a=randint(−10,10)
  ◆◆if a==0:
  ◆◆◆◆a=1
  ◆◆b=randint(−10,10)
  ◆◆x=randint(−10,10)
  ◆◆c=a*x+b
```
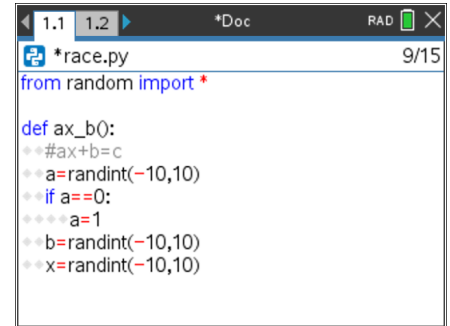
6. Now to create a string display to show the equation.

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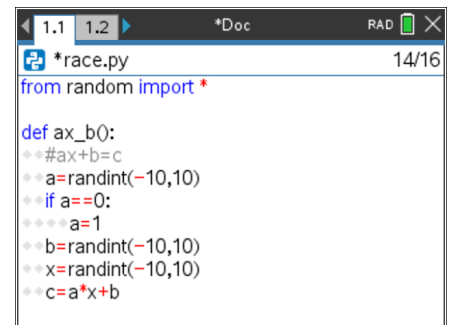
    disp = "x"
    if a == -1:
        disp = "-x"
    elif a != 1:
        disp = str(a) + "x"

    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".

```
◄ 1.1  1.2 ►        *Doc        RAD ▯ ✕
🐍 *race.py                         20/21
  ◆◆◆◆a=1
  ◆◆b=randint(−10,10)
  ◆◆x=randint(−10,10)
  ◆◆c=a*x+b
  ◆◆disp="x"
  ◆◆if a == −1:
  ◆◆◆◆disp="−x"
  ◆◆elif a != −1:
  ◆◆◆◆disp=str(a)+"x"
  ◆◆
```
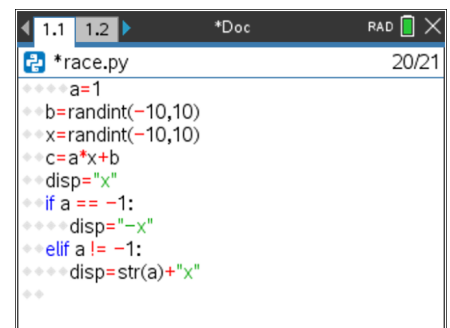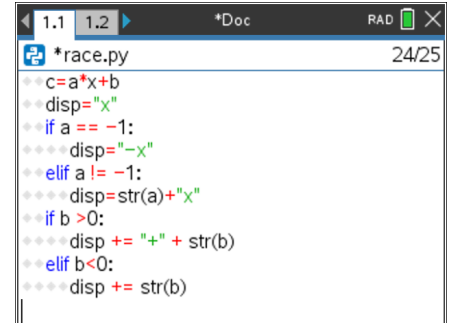
Menu > Built-ins > Control > if
Menu > Built-ins > Control > elif
Menu > Built-ins > Type > str
[ctrl] [=] has both the == and the !=

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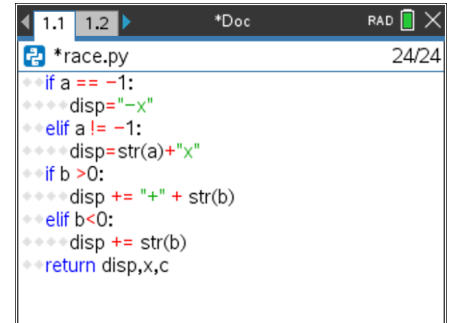
*Make sure you used the += after display. You want to add to the existing display not replace it.

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

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

Menu > Built-ins > Function
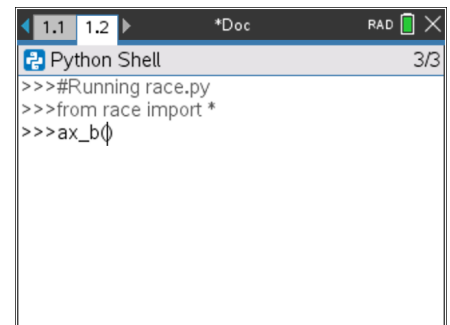
9. Let's try your function out.

Execute your program [ctrl] [r]

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

Press [enter].

The sample to the right displays:
10x – 7,   8,   73

Sample:

That means

10x – 7 = 73…….the value for x should be 8. Let's verify by substitution.

10*8 – 7 =

80 – 7 =

73 = 73

Make sure your values work as well.

We successfully have a function to generate ax+b, x, c

```
>>>ax_b()
('10x−7', 8, 73)
>>>
```

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 type1

Menu > Built-Ins > Function > def function

```
    disp="−x"
elif a != −1:
    disp=str(a)+"x"
if b >0:
    disp += "+" + str(b)
elif b<0:
    disp += str(b)
return disp,x,c

def type1():
    block
```

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?
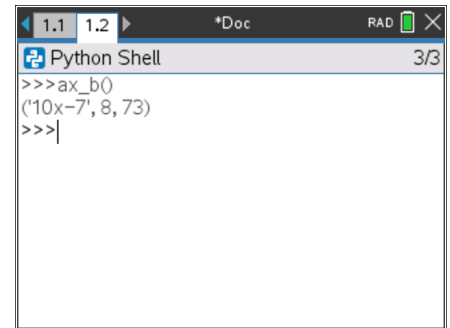
```
    disp="−x"
elif a != −1:
    disp=str(a)+"x"
if b >0:
    disp += "+" + str(b)
elif b<0:
    disp += str(b)
return disp,x,c

def type1():
    disp,x,c=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 "=".
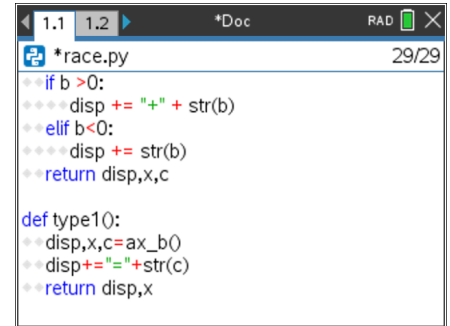
disp += "=" + str(c)
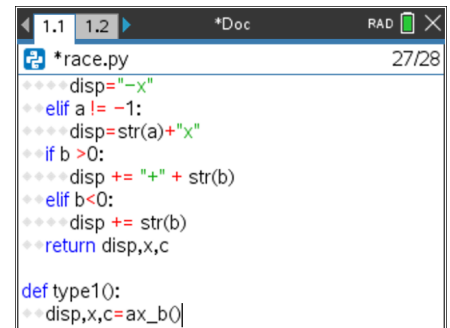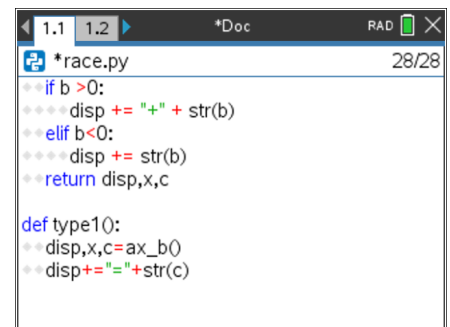
```
if b >0:
    disp += "+" + str(b)
elif b<0:
    disp += str(b)
return disp,x,c

def type1():
    disp,x,c=ax_b()
    disp+="="+str(c)
```

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

    return disp, x

```
1.1  1.2                    *Doc          RAD  X
  *race.py                                29/29
   if b >0:
      disp += "+" + str(b)
   elif b<0:
      disp += str(b)
   return disp,x,c

def type1():
   disp,x,c=ax_b()
   disp+="="+str(c)
   return disp,x
```
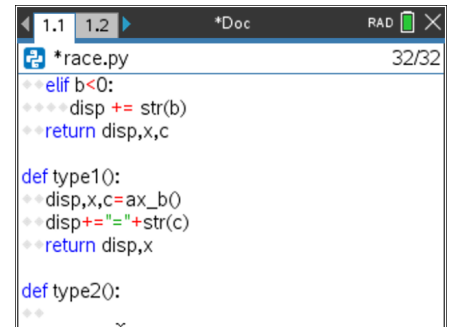
14. Notice your type1 function only has three lines of code. The function ax_b()
    did a lot of the work. Your type2 function will be of the form d(ax+b) = c. It will
    also use the ax_b information to make coding shorter.

    Add a function named type 2.

```
1.1  1.2                    *Doc          RAD  X
  *race.py                                32/32
   elif b<0:
      disp += str(b)
   return disp,x,c

def type1():
   disp,x,c=ax_b()
   disp+="="+str(c)
   return disp,x

def type2():
```
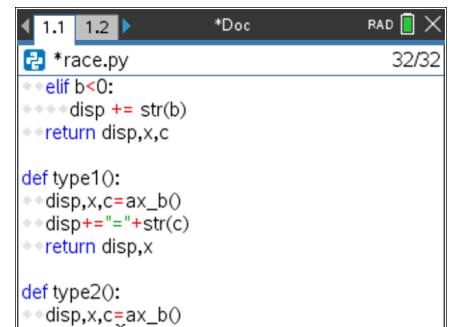
15. Use the ax_b function to generate the left side of the equation, the x and c.

    disp, x, c = ax_b()

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

```
1.1  1.2                    *Doc          RAD  X
  *race.py                                32/32
   elif b<0:
      disp += str(b)
   return disp,x,c

def type1():
   disp,x,c=ax_b()
   disp+="="+str(c)
   return disp,x

def type2():
   disp,x,c=ax_b()
```
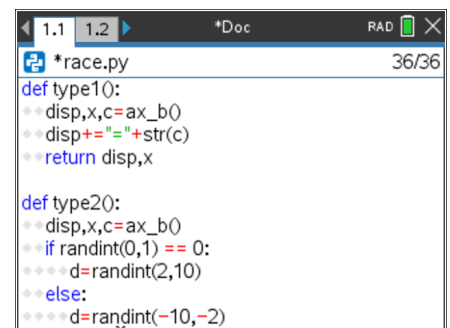
16. This type will be of the form d(ax + b) = c.
    We want d to be between [2,10] or [-10,-2]

    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 Menu > Random > randint

```
1.1  1.2                    *Doc          RAD  X
  *race.py                                36/36
def type1():
   disp,x,c=ax_b()
   disp+="="+str(c)
   return disp,x

def type2():
   disp,x,c=ax_b()
   if randint(0,1) == 0:
      d=randint(2,10)
   else:
      d=randint(-10,-2)
```

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

c = c*d

disp = str(d) + "(" + disp + ")=" + str(c)
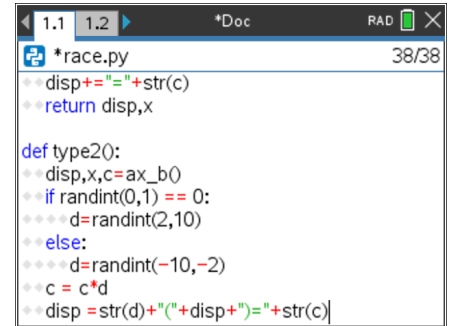
18. Return both the display and x.

return disp, x

19. Using type2, you can easily make a type3 problem that looks like d(ax + b) + e = c.

Create a new function type3

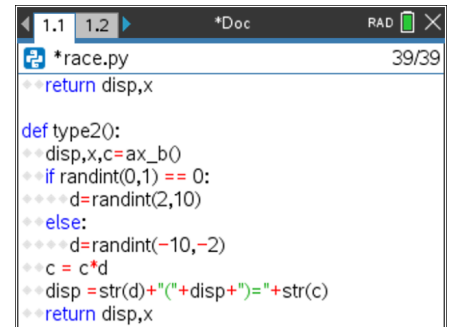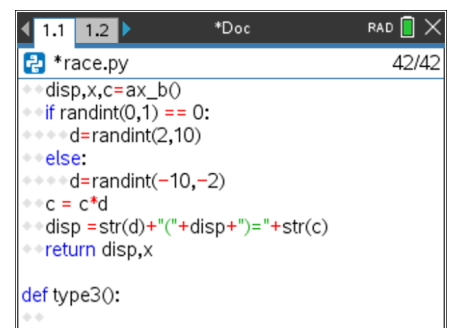20. Put your cursor at the beginning of the first line of the code for type2.

Hold the shift key down while pressing the down arrow to select all the code in the definition. The code should highlight in blue similar to the picture to the left.

Copy the code. [ctrl] [c]

21. Paste the code into type3().   [ctrl] [v]

```
    return disp,x

def type3():
    disp,x,c=ax_b()
    if randint(0,1) == 0:
        d=randint(2,10)
    else:
        d=randint(-10,-2)
    c = c*d
    disp =str(d)+"("+disp+")="+str(c)
    return disp,x
```
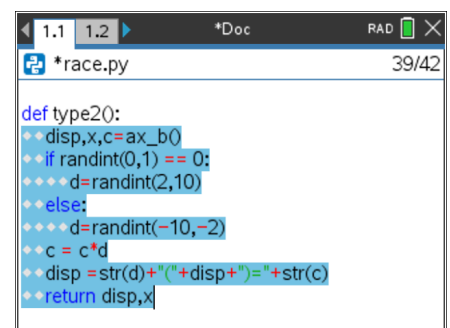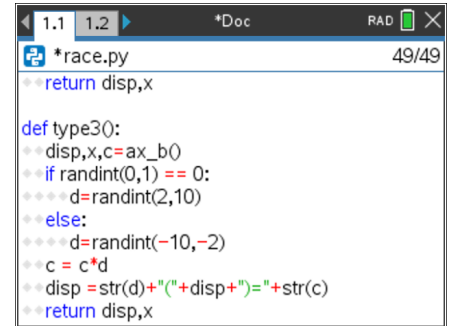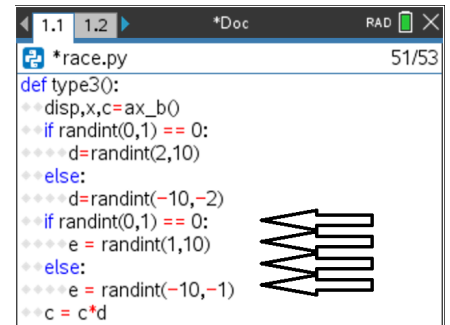
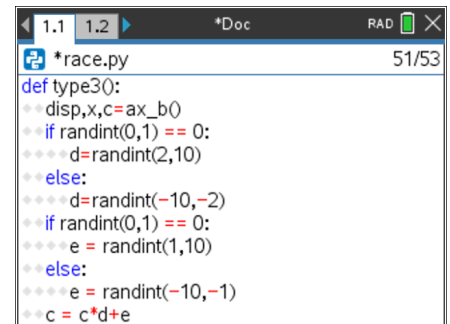22. Insert 4 lines between the if for the d value and the line c = c*d.

    Variable 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)
    else:
        e = randint(-10,-1)

```
def type3():
    disp,x,c=ax_b()
    if randint(0,1) == 0:
        d=randint(2,10)
    else:
        d=randint(-10,-2)
    if randint(0,1) == 0:
        e = randint(1,10)
    else:
        e = randint(-10,-1)
    c = c*d
```

23. Add the e to the total. Change the line c = c*d to
    c = c*d + e

```
def type3():
    disp,x,c=ax_b()
    if randint(0,1) == 0:
        d=randint(2,10)
    else:
        d=randint(-10,-2)
    if randint(0,1) == 0:
        e = randint(1,10)
    else:
        e = randint(-10,-1)
    c = c*d+e
```
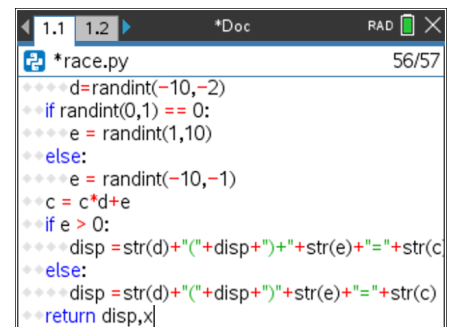
24. Modify the display so it includes e. If e is positive, you'll need to add a "+" as well as the value of e.

    if e > 0:
        disp =str(d)+"("+disp+")+"+str(e)+"="+str(c)
    else:
        disp =str(d)+"("+disp+")"+str(e)+"="+str(c)

```
        d=randint(-10,-2)
    if randint(0,1) == 0:
        e = randint(1,10)
    else:
        e = randint(-10,-1)
    c = c*d+e
    if e > 0:
        disp =str(d)+"("+disp+")+"+str(e)+"="+str(c
    else:
        disp =str(d)+"("+disp+")"+str(e)+"="+str(c)
    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.
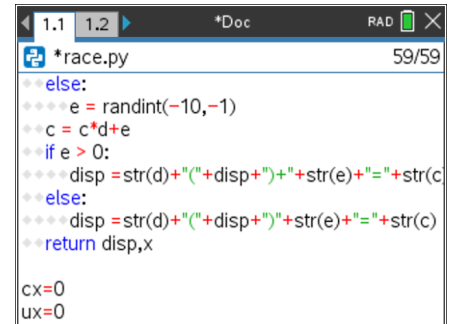   The computer moves 1 to 3 steps.
Display the race

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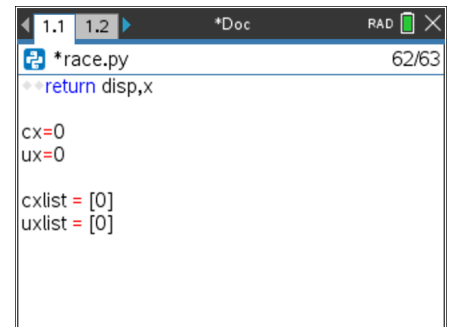Set each one equal to 0.

   cx = 0
   ux = 0

```
else:
    e = randint(−10,−1)
  c = c*d+e
 if e > 0:
    disp =str(d)+"("+disp+")+"+str(e)+"="+str(c
 else:
    disp =str(d)+"("+disp+")"+str(e)+"="+str(c)
 return disp,x

cx=0
ux=0
```

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

   cxlist = [0]
   uxlist = [0]

```
 return disp,x

cx=0
ux=0

cxlist = [0]
uxlist = [0]
```
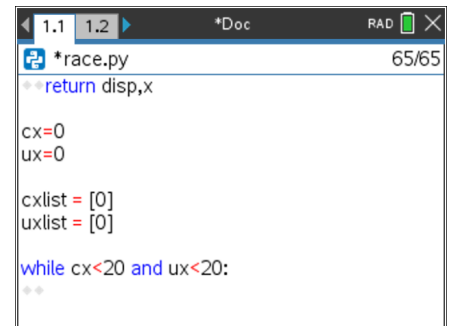
28. While both players have scores under 20, continue play.

   while cx<20 and ux<20:

   Menu > Built-ins > Control > while
   [ctrl][=] and

```
 return disp,x

cx=0
ux=0

cxlist = [0]
uxlist = [0]

while cx<20 and ux<20:
  
```

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

```
n = randint(1,3)
if n ==1:
    disp,x = type1()
elif n ==2:
    disp,x = type2()
else:
    disp,x = type3()
```

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

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

Menu > Control > Type
Menu > Control > I/O

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)
```

**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)
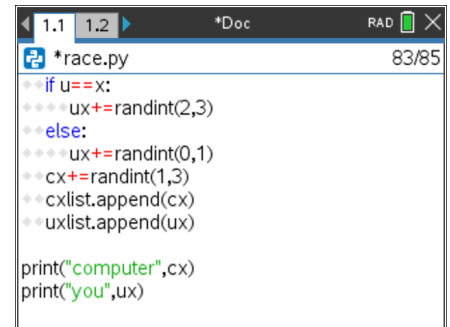cxlist.append(cx)

Menu > Built-ins > Lists > append

```
1.1  1.2  ▶          *Doc        RAD ▢ ✕
 *race.py                          79/81
◦◦◦◦ disp,x=type2()
◦◦ else:
◦◦◦◦ disp,x=type3()
◦◦ u=int(input(disp+", x=  "))
◦◦ if u==x:
◦◦◦◦ ux+=randint(2,3)
◦◦ else:
◦◦◦◦ ux+=randint(0,1)
◦◦ cx+=randint(1,3)
◦◦ cxlist.append(cx)
◦◦ uxlist.append(ux)        I
```

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

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

Menu > Built-Ins > I/O > print

```
1.1  1.2  ▶          *Doc        RAD ▢ ✕
 *race.py                          83/85
◦◦ if u==x:
◦◦◦◦ ux+=randint(2,3)
◦◦ else:
◦◦◦◦ ux+=randint(0,1)
◦◦ cx+=randint(1,3)
◦◦ cxlist.append(cx)
◦◦ uxlist.append(ux)

print("computer",cx)
print("you",ux)
```
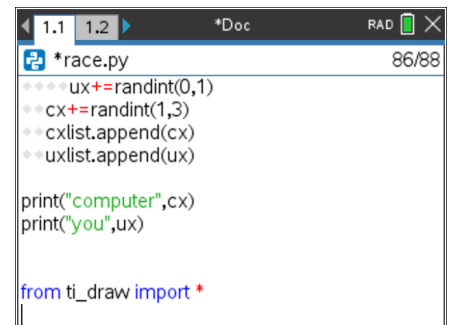
35. Execute your program. [ctrl] [r]

Who won?

Run your program a few more times

36. Now to add the graphics for the race. Add the ti_draw library.
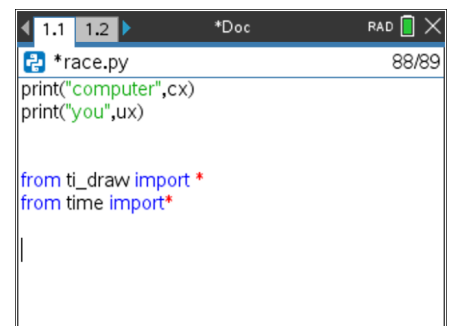
Menu > More Modules > Draw

```
1.1  1.2  ▶          *Doc        RAD ▢ ✕
 *race.py                          86/88
◦◦◦◦ ux+=randint(0,1)
◦◦ cx+=randint(1,3)
◦◦ cxlist.append(cx)
◦◦ uxlist.append(ux)

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

from ti_draw import *
|
```

37. Add the time library.

Menu > More Modules > Time

```
1.1  1.2  ▶          *Doc        RAD ▢ ✕
 *race.py                          88/89
print("computer",cx)
print("you",ux)


from ti_draw import *
from time import*
|
```

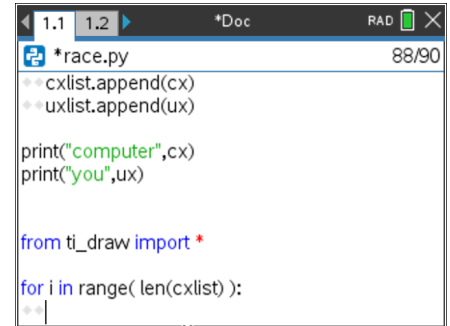38. For each item in the list we need to draw a circle to represent a step.

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

Add the line:
    for i in range(len(cxlist)):

Menu > Built-ins > Control > for
Menu > Built-ins > List > len

```
◄ 1.1  1.2 ►          *Doc        RAD ▯ ✕
🔁 *race.py                        88/90
  ••cxlist.append(cx)
  ••uxlist.append(ux)

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


from ti_draw import *

for i in range( len(cxlist) ):
  ••|
```
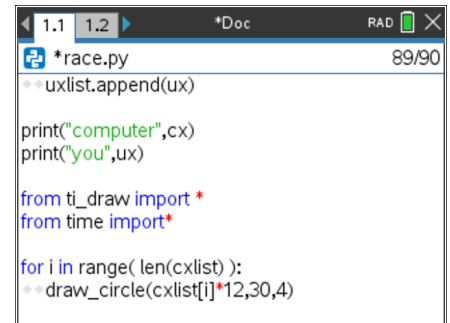
39. Draw the computer's circles using the values in the list.
    Each item in the list can be accessed using the index with square brackets.
    Set the computer y height to 30. Make the circles have a radius of 4.

    draw_circle(cxlist[i]*12, 30, 4)

    Menu > More Modules > TI-Draw > Shape > draw_circle

```
◄ 1.1  1.2 ►          *Doc        RAD ▯ ✕
🔁 *race.py                        89/90
  ••uxlist.append(ux)

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

from ti_draw import *
from time import*

for i in range( len(cxlist) ):
  ••draw_circle(cxlist[i]*12,30,4)
```
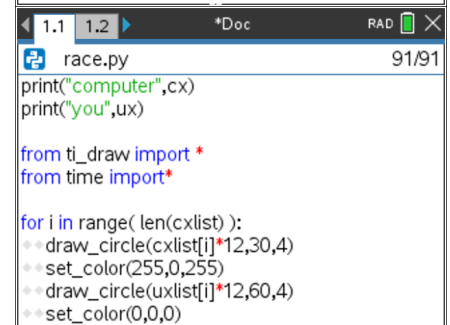
40. Change the color for the user's circles, draw the user's circle, reset the color.

    set_color(255,0,255)
    draw_circle(uxlist[i]*12,60,4)
    set_color(0,0,0)

    Menu > More Modules > TI-Draw > Control > color

```
◄ 1.1  1.2 ►          *Doc        RAD ▯ ✕
🔁  race.py                        91/91
print("computer",cx)
print("you",ux)

from ti_draw import *
from time import*

for i in range( len(cxlist) ):
  ••draw_circle(cxlist[i]*12,30,4)
  ••set_color(255,0,255)
  ••draw_circle(uxlist[i]*12,60,4)
  ••set_color(0,0,0)
```
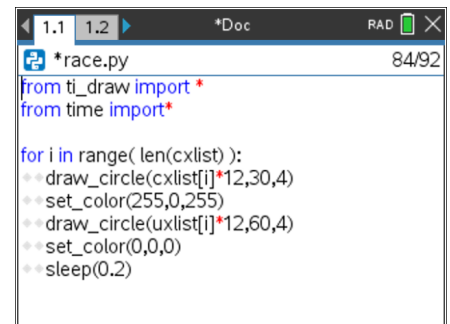
41. Add a short sleep pause for 0.2 seconds.

    sleep(0.2)

    Menu > More Modules > Time > Sleep

```
◄ 1.1  1.2 ►          *Doc        RAD ▯ ✕
🔁 *race.py                        84/92
from ti_draw import *
from time import*

for i in range( len(cxlist) ):
  ••draw_circle(cxlist[i]*12,30,4)
  ••set_color(255,0,255)
  ••draw_circle(uxlist[i]*12,60,4)
  ••set_color(0,0,0)
  ••sleep(0.2)
```
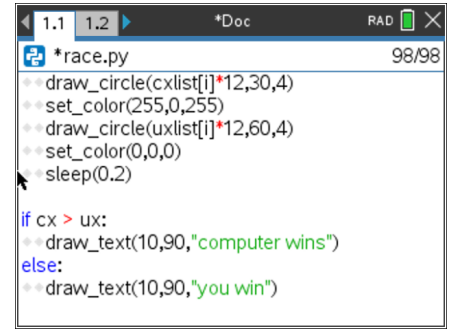
42. Add an if statement that will print the winner.

    if cx > ux:
        draw_text(10,90,"computer win")
    else:
        draw_text(10,90,"you win")

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

44. Challenge:
    Currently, your code from step 42 declares you the winner if you tie.
    Can you modify your code to say "tie", if cx equals ux?