Simulating Fairness: Exploring Probability

In this lesson, you will create a simulation for the game "Evens or Odds". Your simulation will allow you to investigate the fairness of different variations of the game. Along the way, you will learn how to use Python to generate random numbers, store values in a list, use a loop to repeat code, and plot data on a scatterplot. You will use experimental probability to make predictions. You'll use simulations to demonstrate the Law of Large Numbers. Lastly, you will use your simulation and tree diagrams to answer the question, is "Evens or Odds" a fair game under different conditions.

Objectives:

Programming Objectives:

- Use lists to store data
- Use the randint() function to generate random integers.
- Use the plot library to plot points
- Use loops to repeat calculations
- Use if statements to make selections

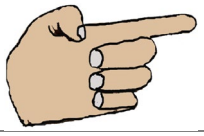
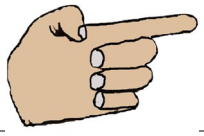
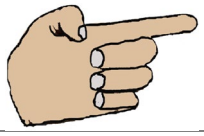
Math Objectives:

- Use simulations and the Law of Large numbers to investigate probability
- Create and use Tree Diagrams to solve problems.

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

In this project, you will synthesize the game "Evens and Odds" and learn some probability along the way. Traditionally, the game is played by two players. One player is "even". The other play is "odd". On the count of three each player holds out either one or two fingers. The sum between the two players determines the winner, "evens" or "odds". Is this a fair game? Does it matter if you pick "evens" or "odds"?

Let's take a look at the game itself.
Player 1 wins if the sum is even, player 2 wins if the sum is odd. Here a few sample plays of the game.

Round	Player 1 Plays:	Player 2 Plays:	Result
1			Sum: 2 Player 1 (evens) wins
2			Sum: 3 Player 2 (odds) wins

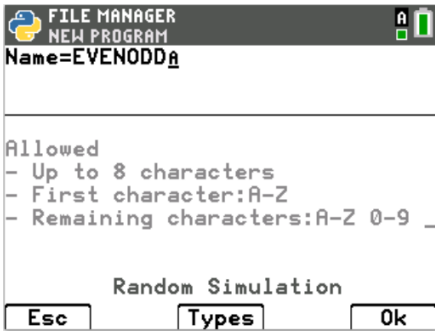
1. If possible, try the game out for yourself. Play the game 6 times with a partner. Record the number of even wins and the number of odd wins.

Evens: Odds:

2. Does the game appear to be fair? If not, which should you pick, "Evens" or "Odds"? Why?

Simulations are often useful to determine if a game is fair. Simulations let you synthesizes thousands of games in a short amount of time. In this project, you will create a coding simulation that will let you investigate the probability of "evens" winning after hundreds of games.

1. The first step will be to create a Python random simulation document.



Create a new python project named "evenOrOdds".

Select "Random Simulations" from the type menu.
This will automatically import the random library.
You will need the randint function from this library.

ti_plotlib
Place your cursor on the line below the from random import *
Fns > Modul > TIPlotLib > **import ti_plotlib as plt**
This library provides the plotting functions.

time
Place your cursor on the line below the from ti_plotlib import *
Fns > Modul > Time > **time import ***
This library provides the sleep function.

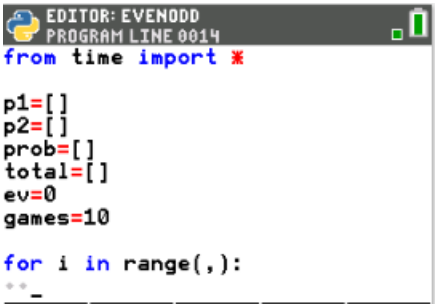
**Tech Tip, Press the [math] button as a shortcut to the Modul menu.

3. You will need 4 lists to keep track of each game's data.
   player 1's selection (1 or 2)
   player 2's selection (1 or 2)
   long run probability of "even" winning, $\frac{number\ of\ even\ wins}{total\ number\ of\ rounds\ played}$
   the total number of games played

   Create the following lists:
   p1 = []
   p2 = []
   prob = []
   total = []

4. Create a variable "ev" to keep track of the total number of evens.
Create a variable named games to set the number of games.
Set ev equal to zero and games equal to 10.

   ev = 0
   games=0



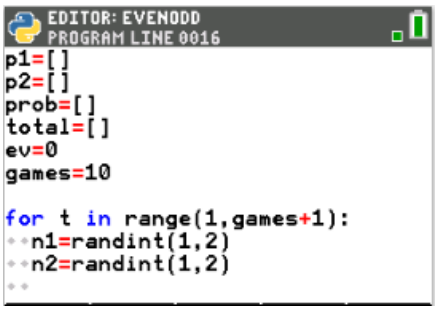
6. Replace the place holder "index" with a variable named t for trial.
   For now, let's play 10 rounds. Let the range start at 1 and stop at games+1.

   It might seem a bit strange to say stop at "games+1" when we want 10 rounds. In Python, the statement **for t in range(1,games+1)** will happen 10 times. Initially, t = 1 will start the loop. Each time through the loop, t increases by 1, then checks to make sure the value is less than 11 before executing the loop.

7. Now to generate and temporarily store each player's random number from 1 to 2. The function randint lets you generate integers.
   Add the lines:
   n1 = randint(1,2)
   n2 = randint(1,2)

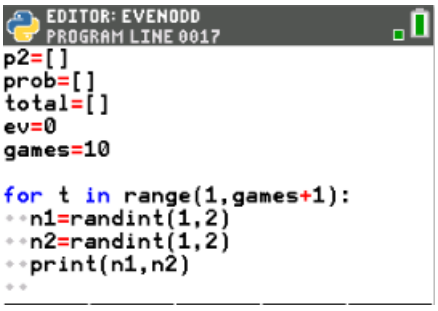
   Fns > Modul > Random > randint

   Make sure these lines are indented two spaces. You should see two diamonds, ◊◊, in front of each line. Python uses indentation to keep the lines of a loop grouped together. If the indentation isn't correct, you'll have an error when you run your program.

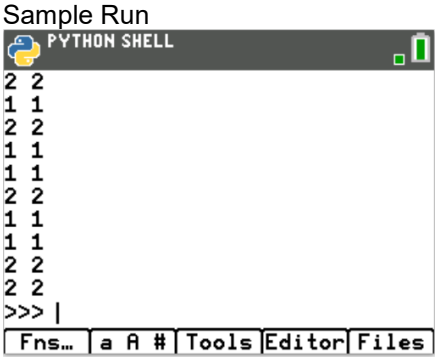


8. It is a good idea to frequently run your program to look for and fix any errors. It is easier to debug often to avoid several bugs from happening at once.

   Add a print statement:
   print(n1, n2)



Sample Run



   Run your program ([Trace]). This should take you to the Python Shell. Did it print 10 random games? Did each player play a 1 or 2?

   A sample of one such run is given to the right.

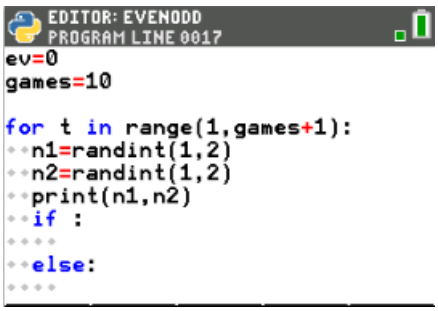
   Run again and it will generate another 10 sets of game play.

   To go back to the code in the Editor, press [Trace}.

9. Now to find the sum and determine if it is even or odd. If the sum is "even", add 1 to the sum total and display even, otherwise display "odd".

   Delete the print statement. Replace it with an if..else statement.

   Fns >Ctl > if..else



10. To determine if a number is even or odd, use the modulo symbol %.
    The modulo symbol determines the remainder after division.

    1%2 = 1 3 %2 = 1 5 %2 = 1

    while

    2%2 = 0 4 %2 = 0 6 %2 = 0

    Thus, if (n1 + n2) % 2 == 0, the sum is even.

    Add this expression to your if statement.
    % and == are both found in [a A #].
    == can also be found in [2nd] [math].

```
EDITOR: EVENODD
PROGRAM LINE 0018
ev=0
games=10

for t in range(1,games+1):
  n1=randint(1,2)
  n2=randint(1,2)
  print(n1,n2)
  if (n1+n2)%2==0:

  else:
```

11. If the sum is even, add 1 to the ev counter variable and print “even”, otherwise print “odd”.

    Notice on the example to the right the two lines below are indented two more spaces.

    ev+=1
    print(n1,n2, “even”)

    If statements use the same indentation rule as the for loops.
    Anything that is part of the if should be indented two spaces.

```
EDITOR: EVENODD
PROGRAM LINE 0022
for t in range(1,games+1):
  n1=randint(1,2)
  n2=randint(1,2)
  print(n1,n2)
  if (n1+n2)%2==0:
    ev+=1
    print(n1,n2,"evens")
  else:
    print(n1,n2,"odds")
```

    Run your code [Trace]. Does it run 10 times? If the sum is even does it print “even”? If the sum is odd, does it print “odd”?

Sample Run:

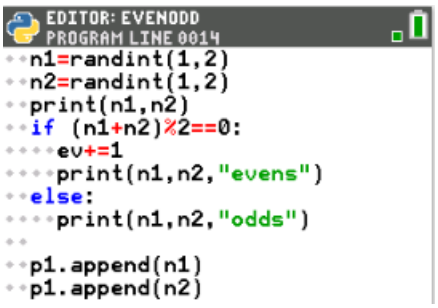
```
PYTHON SHELL
1 1 even
2 2 even
1 2 odd
1 1 even
2 1 odd
1 1 even
1 2 odd
1 2 odd
2 1 odd
2 1 odd
>>> |
Fns… | a A # | Tools | Editor | Files
```

12. Now, add the player's numbers to their lists.
    The function **.append** adds to a list.

    Unindent the next line of code so it only has two spaces.
    The append is part of the loop but not part of the if statement.

    Add the lines:
    p1.append(n1)
    p2.append(n2)

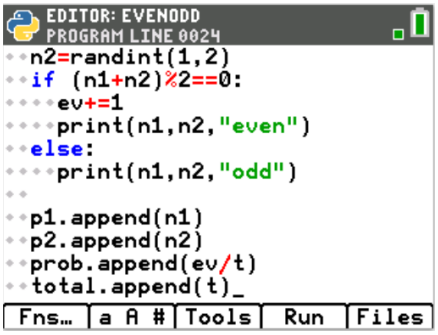
    .append
    Fns > Lists > append



13. Now to calculate the current probability of "evens" winning the game.
    To calculate the probability: $\frac{number\ of\ even\ wins}{total\ number\ of\ games\ played} = \frac{ev}{t}$

    Add this probability to the list of probabilities.
    prob.append(ev/t)

    Add the total number of games played to the list of totals.
    total.append(t)

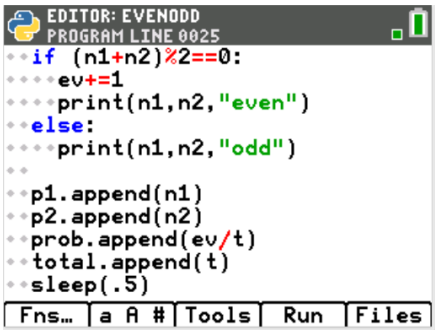


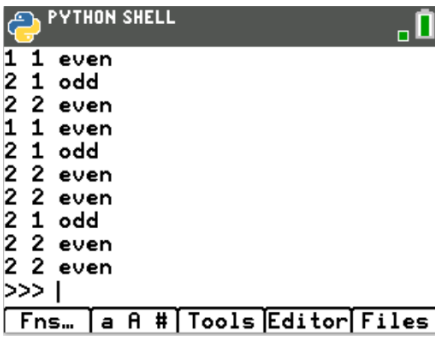
Sample Run

14. Those numbers scroll by awfully fast!
    Add the line sleep(.5) to the end of your loop code.
    This will make the program pause for 0.5 seconds in-between each roll.

    sleep
    Fns > Modul > Time > sleep

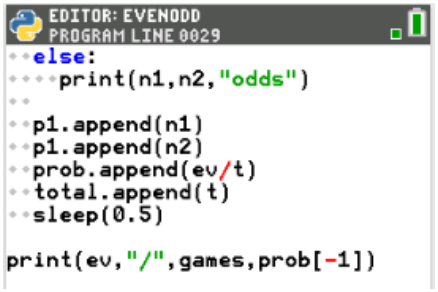
    Run your program. *Do you notice the time difference?*





15. Add a print statement after the loop.
    Make sure to unindent this print statement so it only happens once.

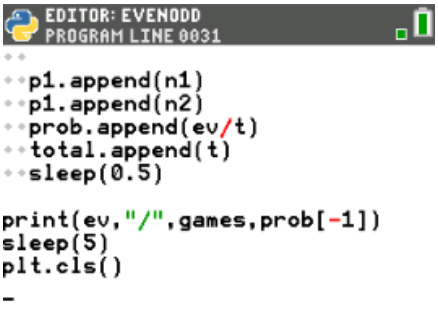
    print(ev, "/", games, "=", prob[-1])



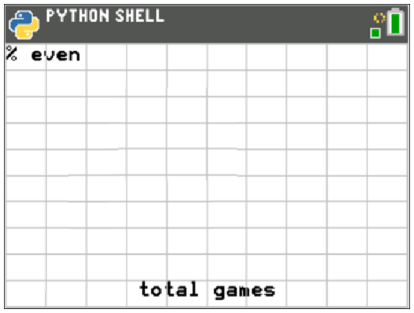
16. The graph will print over the printed statements.
    Therefore, add a sleep statement for a 5 second pause before clearing the print statements.

    sleep(5)
    plt.cls()

    Fns > Modul > TI PlotLib > Setup > clr()



17. Now to set up the graphing window.



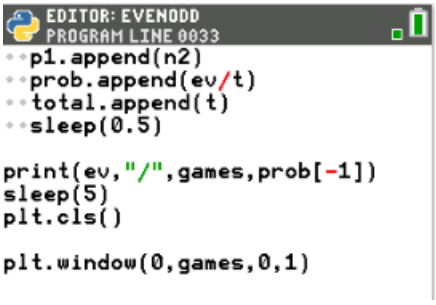
Question 1:
There are a total of 10 games. What do you think we should set the minimum and maximum values for the x -axis?

min x = min y =

max x = max y =

18. To set up the window, you need the window function from the plot library.
Fns > Modul > Ti PlotLib > setup > window

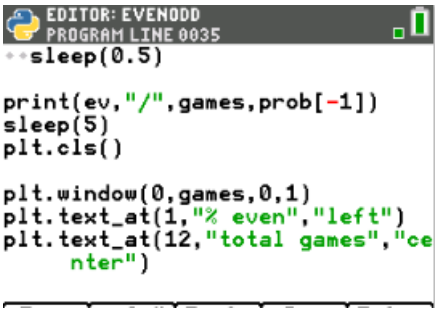
Make the x values go from 0 to games and the y values go from 0 to 1.



19. Now add a label to the x and y axis. The generic function is
Fns > Modul > TI PlotLib > Draw > text_at
plt.text_at(row,"text","align")

You will label the y-axis “% even”. The y-axis is on the left of the screen. Therefore, add the line
plt.text_at(1,"% even","left")

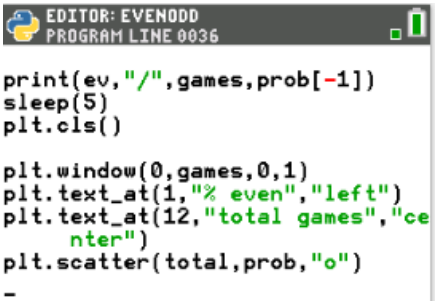
You will label the x-axis “total games”. The x-axis label will be in the center. Therefore, add the line
plt.text_at(12,"total games","center")



20. Plot the data.
Fns > Modul > TI PlotLib > Draw > Scatter

The x-list is the total list. Your y-list is prob.

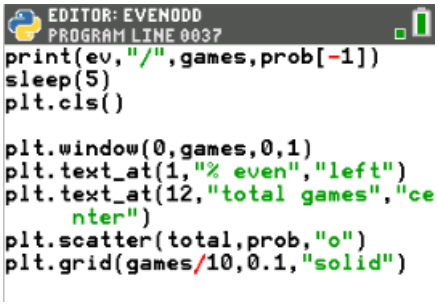
plt.scatter(total, prob, “o”)



21. Your graph is missing some grid lines.
Fns > Modul > TI PlotLib > Setup > grid

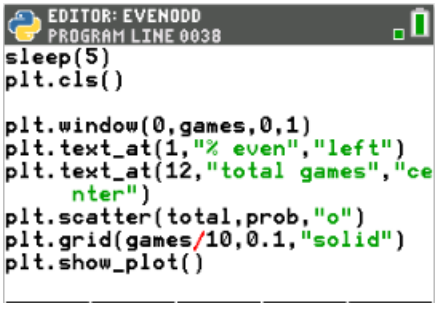
Since the x’s have a range of [0,10], the x step should be 1.
The y’s have a range of [0,1]. Make the y step 0.1

plt.grid(games/10, 0.1, ”solid”)

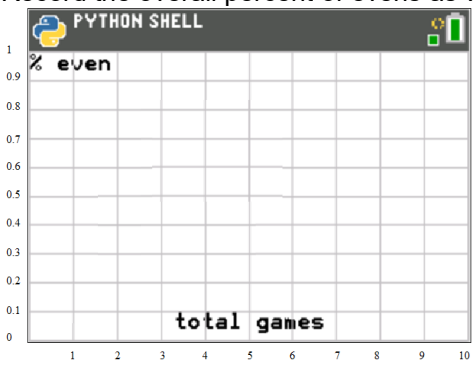


22. Tell the program to show your plot.
plt.show()

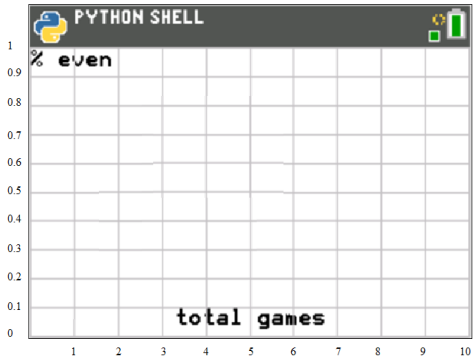
Fns > Modul > Ti PlotLib > Draw > show



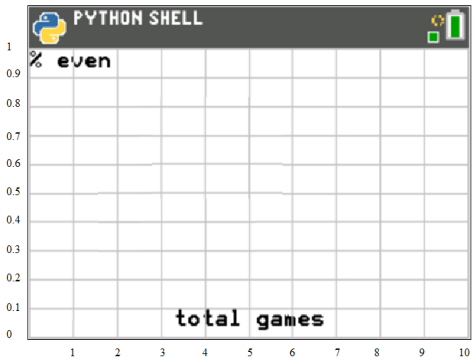
23. Run your program 4 times. Each time you execute your program, copy the graph onto a new graph below. Record the overall percent of evens as well.



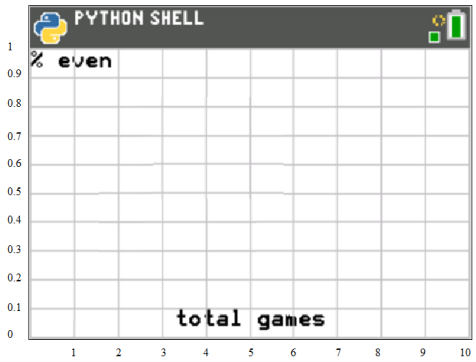
$\frac{\quad}{10} \approx$



$\frac{\quad}{10} \approx$



$\frac{\quad}{10} \approx$



$\frac{\quad}{10} \approx$

Do you notice any patterns to your data?

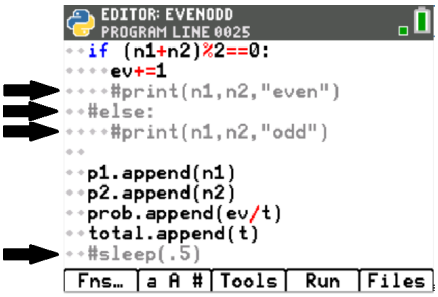
According to your graphs, is the game fair? If not, which should you pick, evens or odds?

What do you think would happen if you played 250 games instead of 10?

What would the graphs look like?

What proportion of games would evens win?

24. Let's change your code to run 250 times.



First, let's remove the print statements and the sleep(0.5).
If we leave them in, you'll have to wait 250x0.5 = 125 seconds.
That's more than 2 MINUTES!

In front of the two **print** statements add a # symbol.
This makes those two lines comments. The computer will skip them.

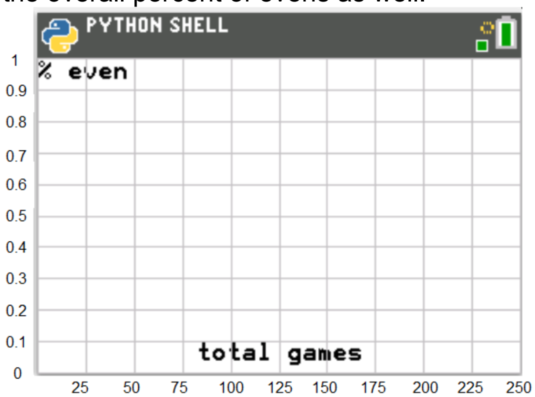
Add a # to the else statement as well.

Finally, put a # infront of the sleep command.

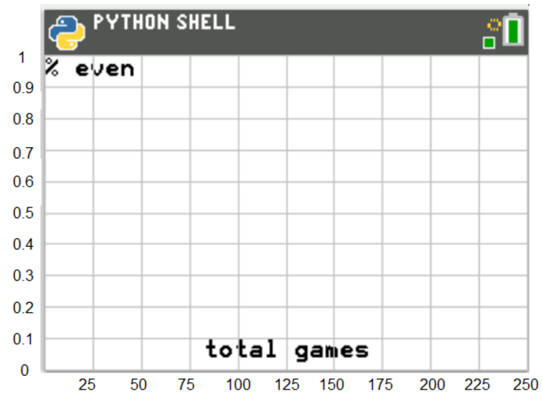
To add the # symbol, put the cursor at the beginning of a line [a A #].

25. Next, change games from 10 to 250

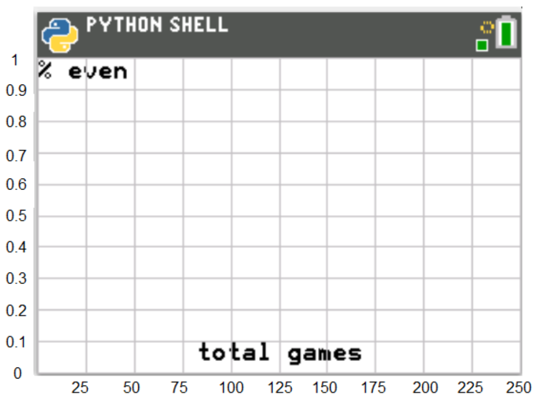
26. Run your program 4 times. Each time you execute your program, copy the graph onto a new graph below. Record the overall percent of evens as well.



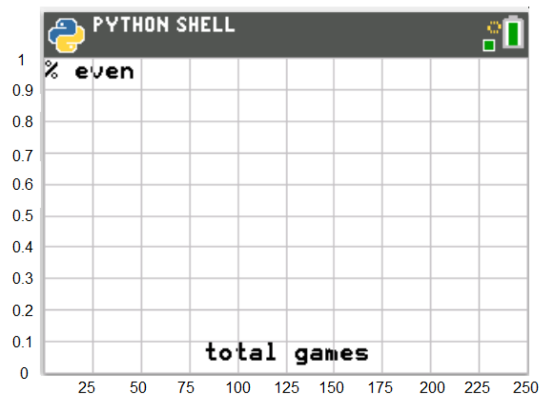
$\frac{\quad}{250} \approx$



$\frac{\quad}{250} \approx$



$\frac{\quad}{250} \approx$



$\frac{\quad}{250} \approx$

Do you notice any patterns to your data?

What do you think would happen if you played 2500 games instead of 250?

What would the graphs look like?

What proportion of games would evens win?

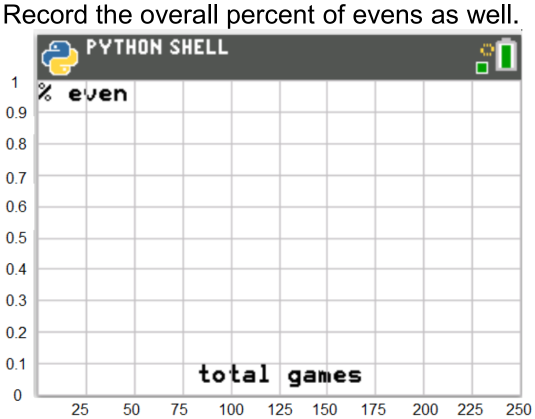
The graphs above illustrate **The Law of Large Numbers**. The Law of Large Numbers states that the more games you play, the long-term probability will approach the theoretical probability. In this case, the theoretical probability "evens" will win the game is 0.5. Therefore, it doesn't matter if you pick "evens" or "odds", the probability of winning a game is 0.5 regardless your pick.

27. What if we change the rules of the game? What if you can play either a one, two, or a three? Would the game still be fair? Make a prediction. Do you think this new version would be fair? Why?

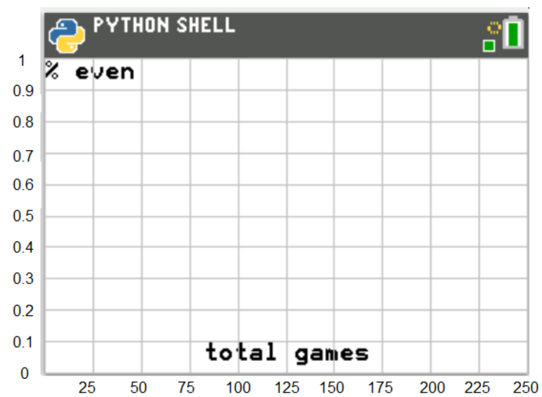
28. Modify your game. The players now can play a one, two, or a three.

    n1 = randint(1,3)
    n2 = randint(1,3)

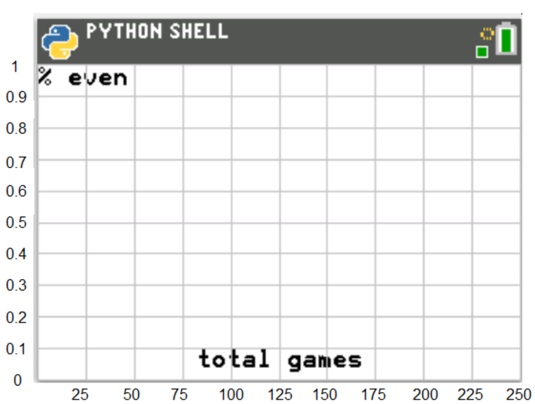
29. Run your **NEW** program 4 times. Each time you execute your program, copy the graph onto a new graph below. Record the overall percent of evens as well.



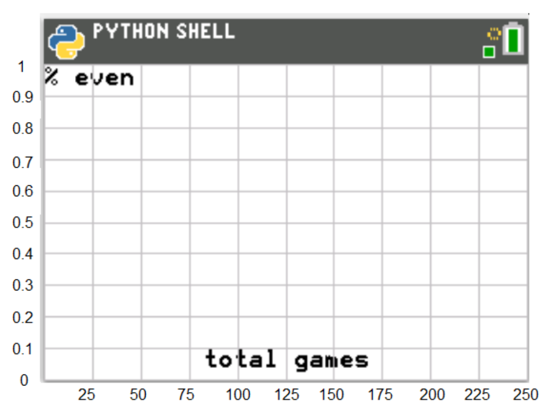
$\frac{\quad}{250} \approx$



$\frac{\quad}{250} \approx$



$\frac{\quad}{250} \approx$

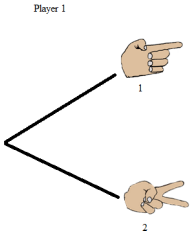


$\frac{\quad}{250} \approx$

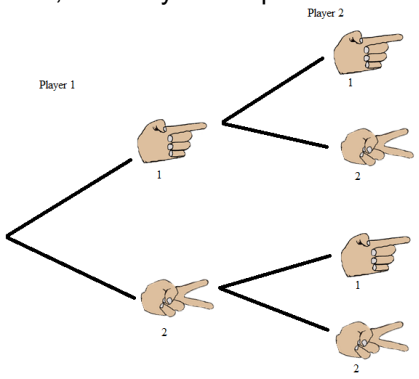
This version of the game is NOT fair. Why not? What does the theoretical probability of "evens" winning the new version?

30. How could we determine the probability without a simulation? One way would be to create a **tree diagram** that represents all the possible ways to play the game.

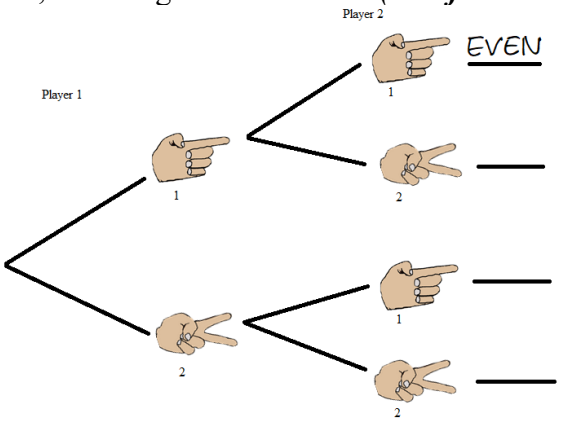
Step 1, Draw the possibilities for Player 1



Now, add Player 2's options.



Last, add the game outcomes. ***(The first one is done for you. Add the other three outcomes.)***



34. Use the tree diagram above to answer the following questions:

How many possible outcomes are there? _____

p(sum even) = p(sum odd) = p(sum is 3) = p(sum < 4) =

Does the probability of evens match your graphs from step #28?

35. The graphs in step 31 showed the NEW game was not fair if the players were allowed to play a one, two or three. Let's investigate why this version isn't fair.

a. Create a tree diagram for this version of the game. You don't have to draw the physcial hands like the example above, you may choose to only put the numbers at the end of the line segments.

b. How many outcomes are possible in this version?

c. The game wasn't fair. Use the tree diagram to give the probability "evens" wins.

d. p(sum odd) = p(sum is 3) = p(sum < 4) = p(sum > 6) =

36. What would happen if:

player 1 could only play a 1 or a 2
player 2 could play a 1, 2 or 3

a. Create a tree diagram for this version of the game.

b. Is this version of the game fair? Explain using your tree diagram.

s