End of Course Projects

After working through all of the Units 1-5 (and maybe Unit 6 if you have CX II) in TI Codes, you now have a toolbox full of tricks, and might be surprised at what you can do!

Objectives:

- Try these additional tasks to practice what you learned, and combine your skills and commands in new ways!

---

1. **Divisibility** in a program is determined by any one of the following conditions:

   - **If x/a = int(x/a)**
   - **If fpart(x/a) = 0**
   - **If mod(x,a) = 0**

   Try writing the following collection of programs *(each follows from its predecessor)*:

   a. Input two numbers and tell whether the first is evenly divisible by the second.

   b. Input a number and use a loop to display all the divisors of the number.

   c. Input a number and use a loop to *count* the number of divisors of a number. Display the count.

   d. Input a number and state whether the number is prime. (**Prime** numbers have exactly two divisors, 1 and itself. 1 is not prime).

   e. Use nested loops to display prime numbers (Input an upper limit).

   f. Modify the prime number program from e. to allow the user to input a lower and an upper limit.

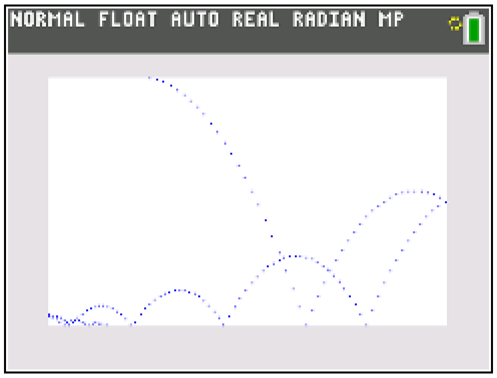
   g. Input a number and display the *sum* of its *proper* divisors (do not include the number itself). Can you make the search for divisors more efficient? Hint: How large is the largest proper divisor of a number? Can you do better?

   h. Tell whether an Input number is a **Perfect Number** (a Perfect number equals the *sum* of its *proper* divisors).

   i. Find some **Perfect** numbers. Try using a lower and upper limit. Start with 1..500.

   j. Amicable, or Friendly Numbers are two numbers in which each equals the sum of the other's proper divisors. Find the pair of Amicable Numbers below 1000.

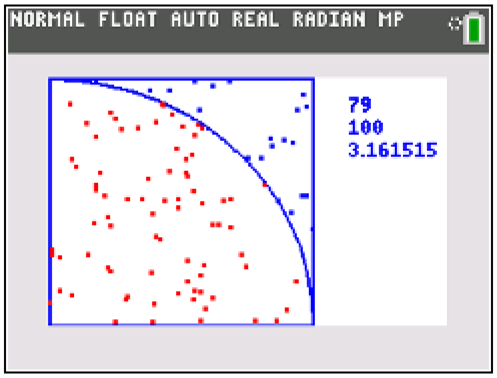
2. **The Bouncing Ball** (physics): A ball is given an initial horizontal and vertical velocity and falls due to the force of gravity. Write a simulation of the bouncing ball. When the ball hits the side of the screen it reverses direction. When the ball bounces it loses some vertical velocity.



(Demo uses Pxl-On or Pt-On for speed.)

3. Simulation: **Toss Two Dice** using **randInt(1,6)** and keep track of the average sum. Bonus: Keep track of (*count*) the occurrence of each sum (2..12) in a list (how many 2's, 3's, 4's,…,12's) and make a histogram using the Stat tools in the calculator of those totals. Make list[1] equal to 0.

4. Simulation: **π darts**: Imagine a unit quarter circle (radius 1) at the origin and a unit square around it. Toss a dart at the dartboard using **rand** from the **[math] PROB** menu. **rand** gives a random decimal value between 0 and 1. Use **rand→X** and **rand→Y**. Determine whether the point (X,Y) is inside the circle using the distance formula. If it is, it counts ad a 'hit'. Keep track of both the hits and the total number of tosses. These points are all going to be in the First Quadrant only so display the value 4*hits/total. Why 4*? Bonus: Use Draw commands to display the darts hitting the board and display the approximation on the drawing as well while the program is running.



(note that 4 x 79 / 100 = 3.161515)

5. **Space Travel.** Write a program to input a weight and display all the weights on other planets (including Earth and Pluto) and the moon. You will have to look up the conversion factors online. Use either **Output( )** or **Text( )** to display all the values at once.

6. **Guess My Number**. The computer chooses a random number from 1 to 100. The user must guess the number. The computer tells the user whether the guess is too high or too low. The game ends when the user guesses the number correctly.

7. **The Game of Bagels.** The computer picks a 3-digit number at random in which no two digits are the same. The user must guess the number. If the human has a correct digit in the correct position then the computer displays "Fermi". If the human has a correct digit in the wrong place then the computer displays 'Pico'. If none of the human's digits are in the computer's number then the computer displays 'Bagels'. Keep track of the number of guesses it takes for the human guess the computer's number.

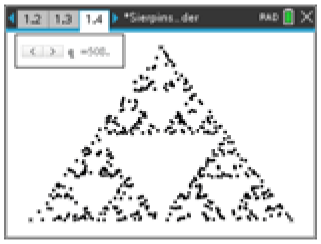
8. Simulation: **Random Chords**. Pick two points on a unit circle. What is the average length of the chord?

9. Simulation: **Random segment in a square**. Pick two random points in a unit square. What is the average distance between them?

10. The **Fibonacci** sequence:
    a. Write a program to generate a list of Fibonacci numbers: 1, 1, 2, 3, 5, 8, 13… in which each number is the *sum* of the previous 2. Start with the **list:={1,1}** and build the rest.
    b. Determine the *ratios* of each element to its previous neighbor (1/1, 2/1, 3/2, 5/3, 8.5, 13/8, …) expressed as a decimal. What value do the ratios approach in the long run?
    c. Create a list of the *reciprocals* of the Fibonacci numbers: 1, 1, 1/2, 1/3, 1/5, 1/8, 1/13… and determine the sum. Is it finite or infinite if you continue?
    d. Modify your program to accept the first two values as arguments. Do the same patterns (in b. and c.) appear regardless of the starting numbers?

11. **Sierpinski's Gasket Chaos Game** *(requires a program, Graph with scatter plot, a slider, and the program running in a Math Box on a Notes app)*

    Write a program **sier(n)** to make two lists (**xs** and **ys**) that store coordinates of points and plots them as a scatter plot on a Graphs app.



12. Write a program that uses **Draw** statements to create a Sierpinski Gasket using the above algorithm!

13. **Random Walks**:

    a. Start at (0, 0) and move left or right one unit at a time either left or right. After 100 moves, how far are you from (0, 0)? Design a graphical simulation.

14. b. Suppose you start at (0,0). Choose a direction to move in (up, down, left, or right) and move one unit in that direction. How long will it take to get to (7,3)? Create a graphical simulation.

15. Write a program to **Draw** an emoji. Start with the original, the smiley face:

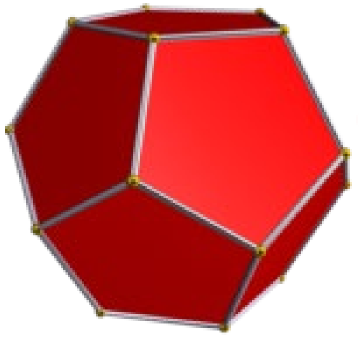
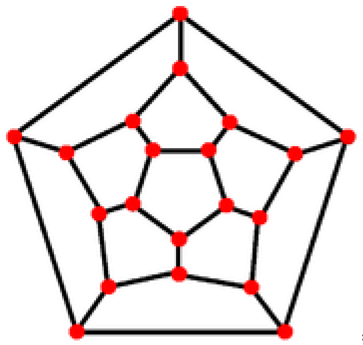
    … then create your own. You could be famous!

16. Hunt the Wumpus! (a text-based game originally developed by Gregory Yob in 1973)

In this game the player is in a cave system consisting of 20 numbered rooms connected by tunnels. Each room leads to two others. In one of the rooms is a "Wumpus", which the player is attempting to kill. Two of the rooms contain bottomless pits. Another contains "super bats" which will pick up the player and transport her to a random cave. You are also carrying a bow and five arrows. On your turn you are told your room number and the numbers of the two connected rooms. You are then asked to either shoot or move and into which room. If you choose to shoot then you choose one of the two rooms to shoot into (if you have arrows left). You might shoot the Wumpus. You might scare the bats to fly to another room. One room contains 'treasure' and one room contains the 'exit'. If you choose to move then you are asked which of the two available room to move to. If you *locate the treasure and find the exit* you win the game. If you meet the Wumpus or fall into a bottomless pit you lose the game. If you meet the “bats” then they transport you to a random room (possibly one containing the Wumpus or a pit!).

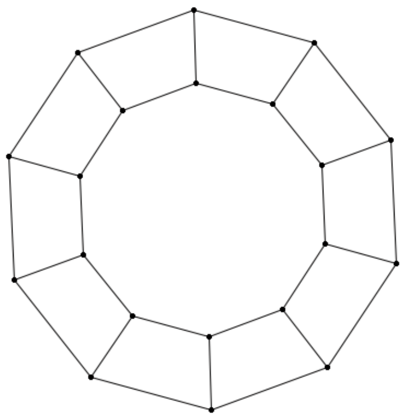
Optional: enable the Wumpus and the bats to move from room to room as the player moves. The Wumpus and bats are not affected by the treasure, pits or exit.

One conception for the cave system is a dodecahedron (12 faces and 20 vertices):

*View When flattened)

Another is two concentric rings of 10 rooms each that are connected to each other and to the one nearest it on the other ring (you can number the vertices of either the pentagons (above) or the decagons (below) from 1 to 20):



Remember to share your success on social media and tag us #TICalculators