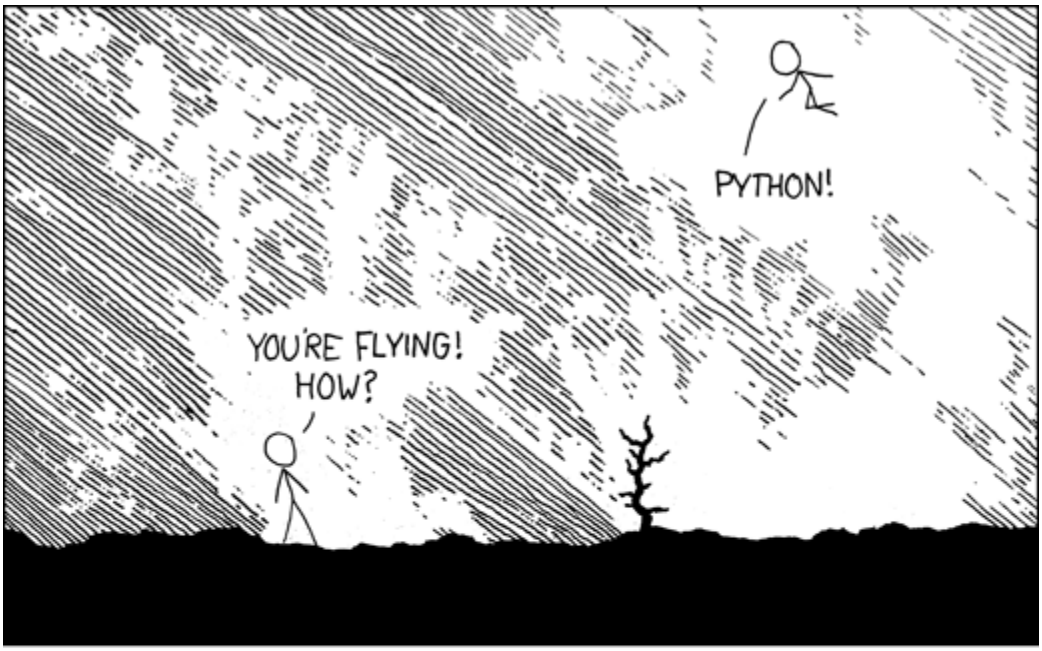
# TI-84 Plus CE Python: Sample Programs

Webinar, Nov 16, 2021

John Hanna, johnehanna@gmail.com
Becky Byer, bbyer31415@gmail.com
with guidance from Steve Phelps

No program is ever finished.

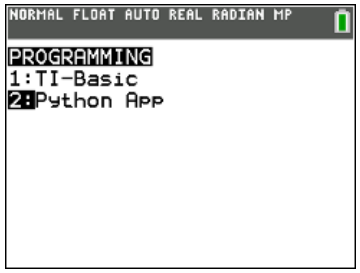




Permanent link to this comic: https://xkcd.com/353/

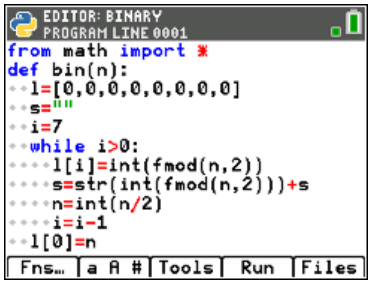
# The Big Picture

The **TI-84 Plus CE Python Edition** is a *new* graphing calculator that supports Python programming using the 'Python App' . Pressing the **[prgm]** key now provides *two* options:

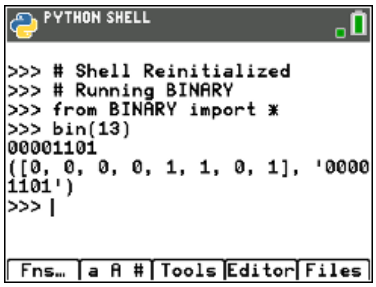



Selecting the **Python App** takes you to the Python programming experience which is a *separate* system from the rest of the graphing calculator. There are three workspaces in the Python App:


**File Manager**




**Editor**




**Shell**

**File Manager:** where new Python files are created and allows you to **<Manage>** files (copy, rename, delete)

**Editor:** where Python code is written. **<Tools>** contains editing utilities like Copy line and Paste line.

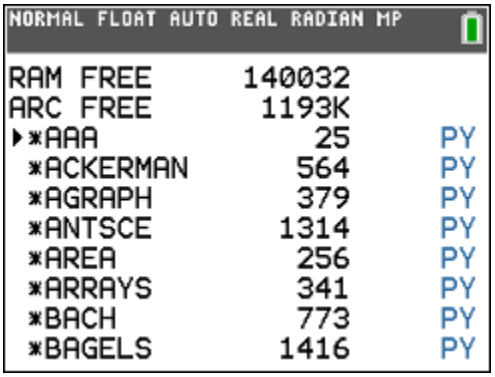
**Shell:** where Python programs run. Similar to the calculator Home screen, also allows for evaluating expressions and testing short code chunks.

**Soft Keys**: all three workspaces have 'soft keys' on the bottom of the screen. Use the top row of keypad keys to access these soft keys.

**Memory Management**

Python files are stored as **AppVars**.

Active (available) files are stored in RAM. Backup files are stored in ARCHIVE (*).




**[quit]** the Python App.

Press **[mem] > Memory Management > AppVars**

Python files are indicated by the **PY** on the right.

Press **[enter]** to move a file between RAM and ARCHIVE.

Note the asterisks (*) indicating the archived files.

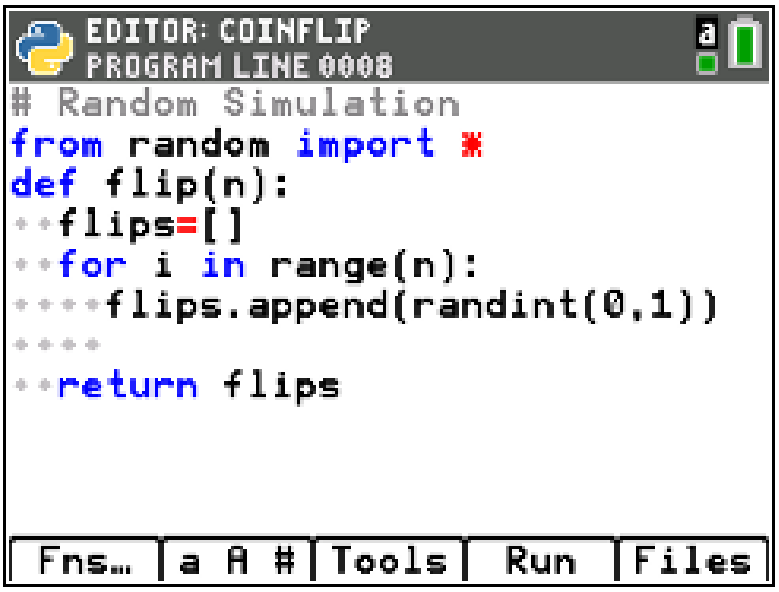
In the Python App you are limited to 100 Python files and 50k RAM so archiving becomes important. Archived files will not appear in the Python App File Manager.

**Editing Tips**

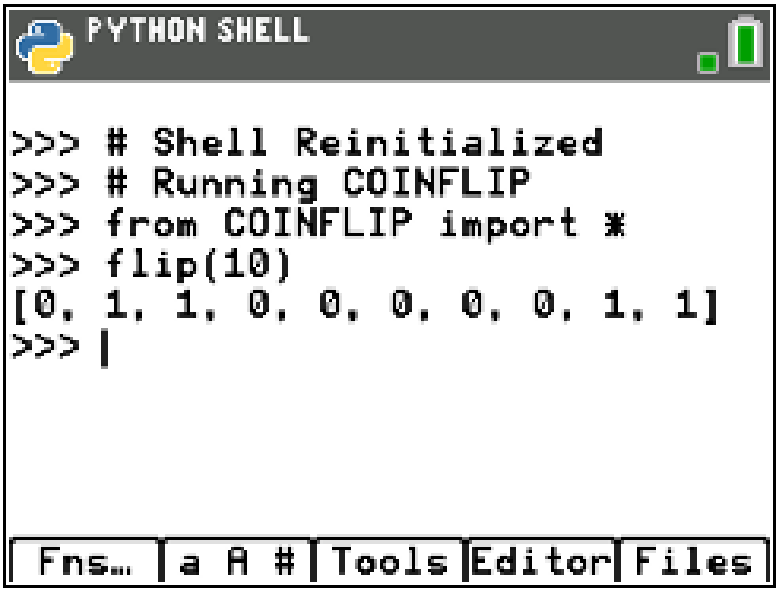
Use the menus. Use short variable names. Indentation is crucial! Python is case-sensitive: use **[alpha]** or **[2nd] [alpha]** and pay attention to the on-screen prompts and cursor for the current state (lower, upper, or numeric). The **[math]** key is a shortcut for **<Fns…> Modul**.

# Sample functions and programs

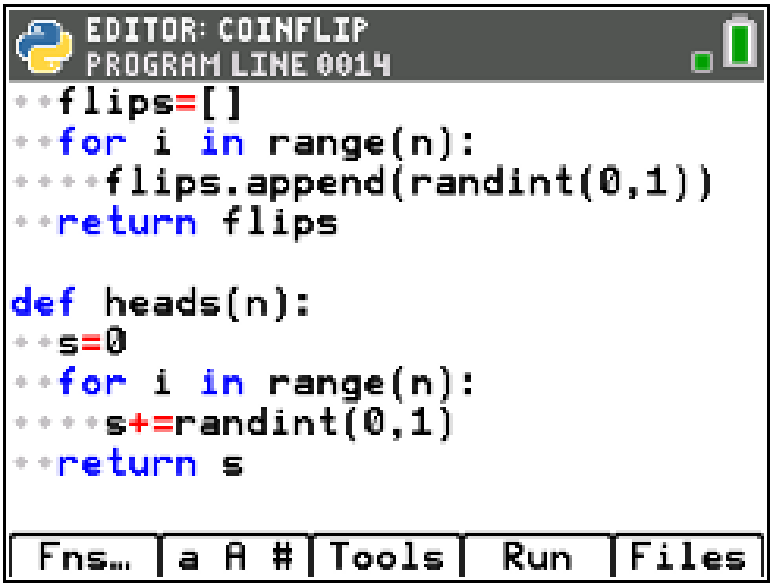
### A Coin-Flipping Function



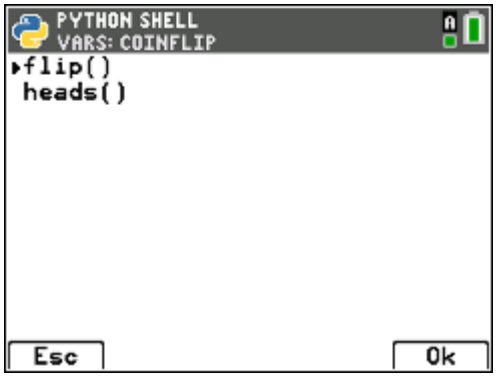

select **<Run>**... then press **[vars]** and select **flip(**10**)** (type the 10)



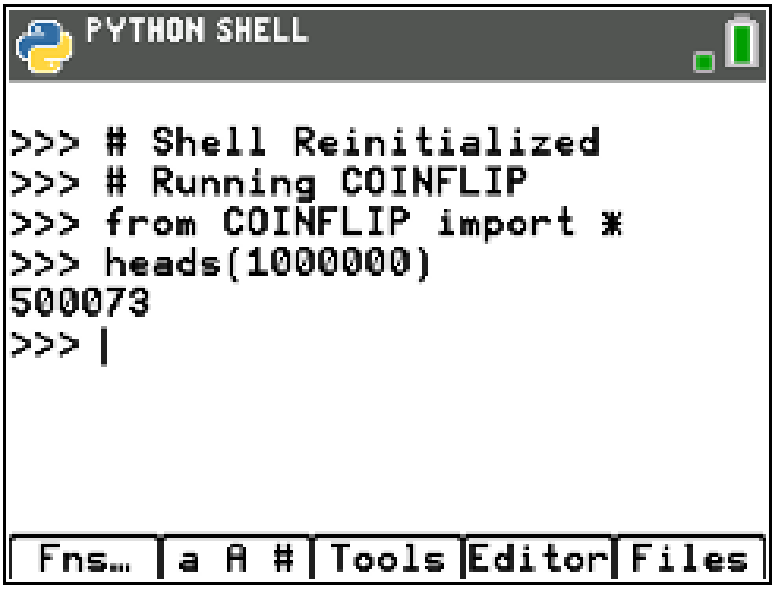
**How Many Heads? Function**
(note: same file!)



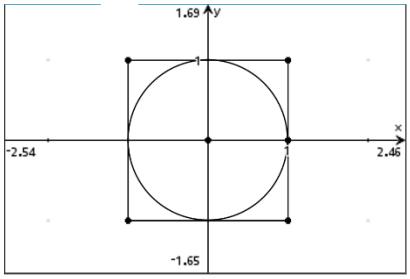

press **[vars]** to see this...




then select **heads()** from the list...



## Pi Darts Program

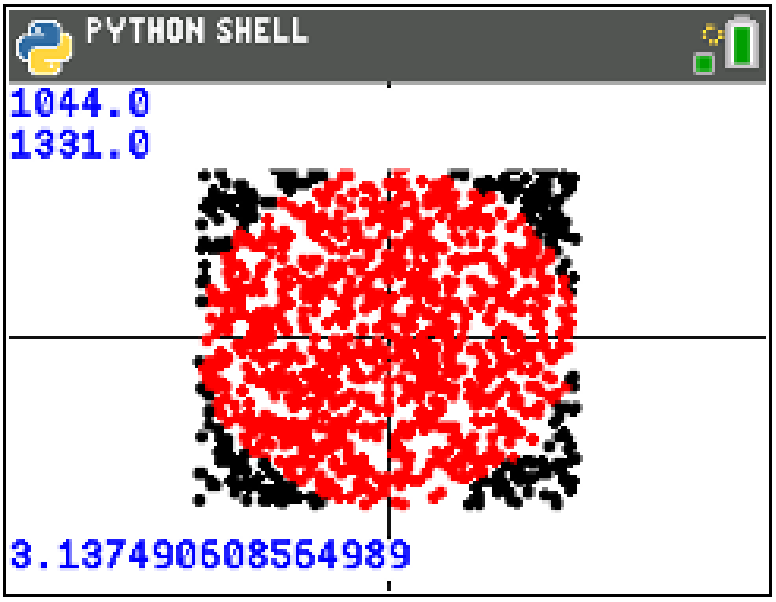



Darts are thrown randomly at the unit circle inscribed in a 2x2 square. Some land inside the circle, others don't. On average, what percent of the darts land inside the circle? How can we use this to compute an estimate for pi?

```
EDITOR: PIDART
PROGRAM LINE 0005
# Random Simulation
from random import *
from ti_system import *
h=t=0
while not escape():
  t+=1
  x=uniform(-1,1)
  y=uniform(-1,1)
  if x**2+y**2<1:
    h+=1
  print(4*h/t)
Fns… | a A # | Tools | Run | Files
```

**PIDARTS: A graphical version:**

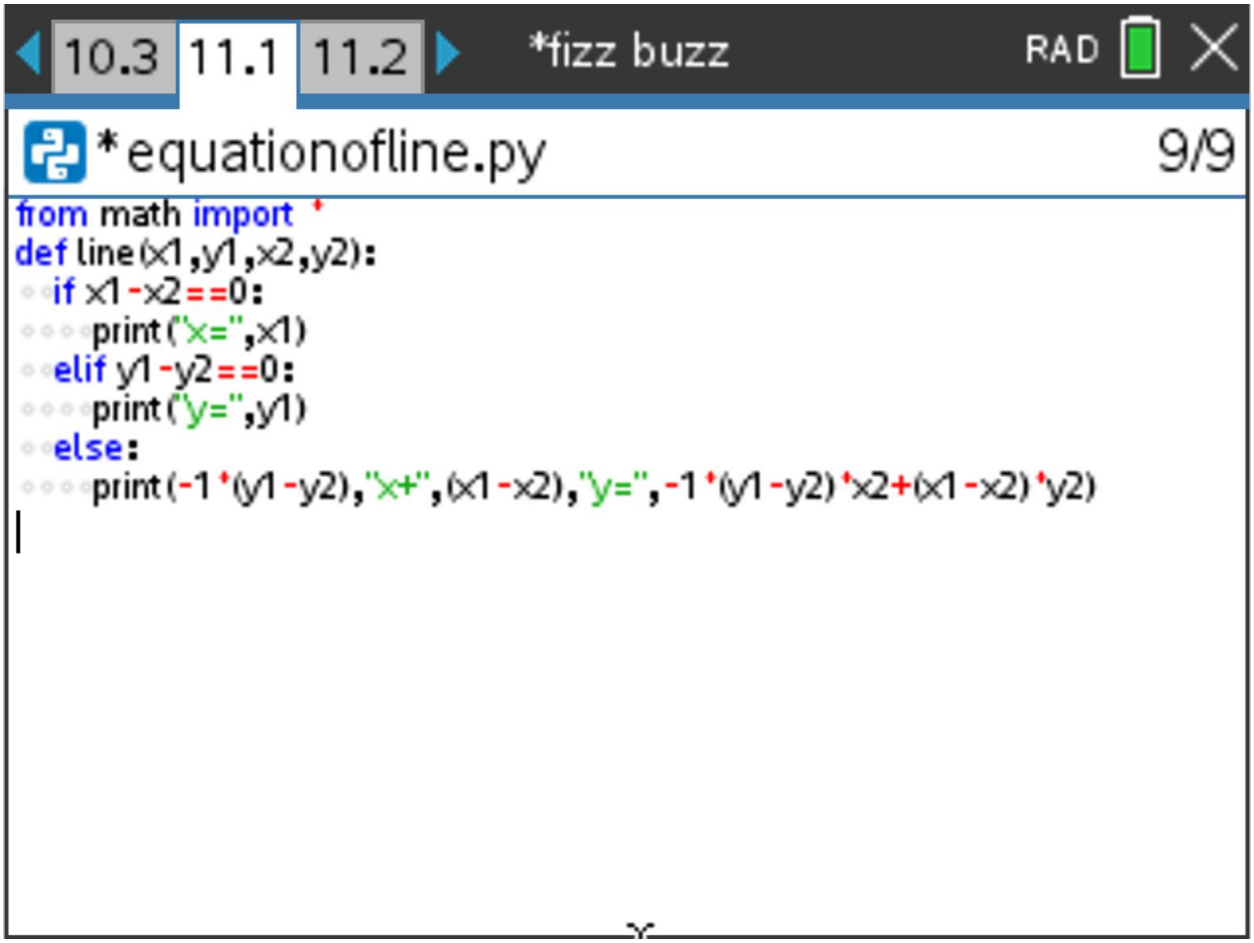
```
from ti_system import *
from random import *
import ti_plotlib as plt

plt.cls()
plt.window(-2,2,-1.5,1.5)
plt.axes("axes")

try:
  l=recall_list("DARTS")
  hits=l[0]
  total=l[1]
except:
  hits=0
  total=0

while not escape():
  total+=1
  x=uniform(-1,1)
  y=uniform(-1,1)
  plt.color(0,0,0)
  if x*x+y*y<=1:
    hits+=1
    plt.color(255,00,0)
  plt.plot(x,y,"o")
  plt.color(0,0,255)
  plt.text_at(2,str(total),"left")
  plt.text_at(1,str(hits),"left")
  plt.text_at(12,str(hits/total*4),"left")

plt.show_plot()
store_list("DARTS",[hits,total])
```



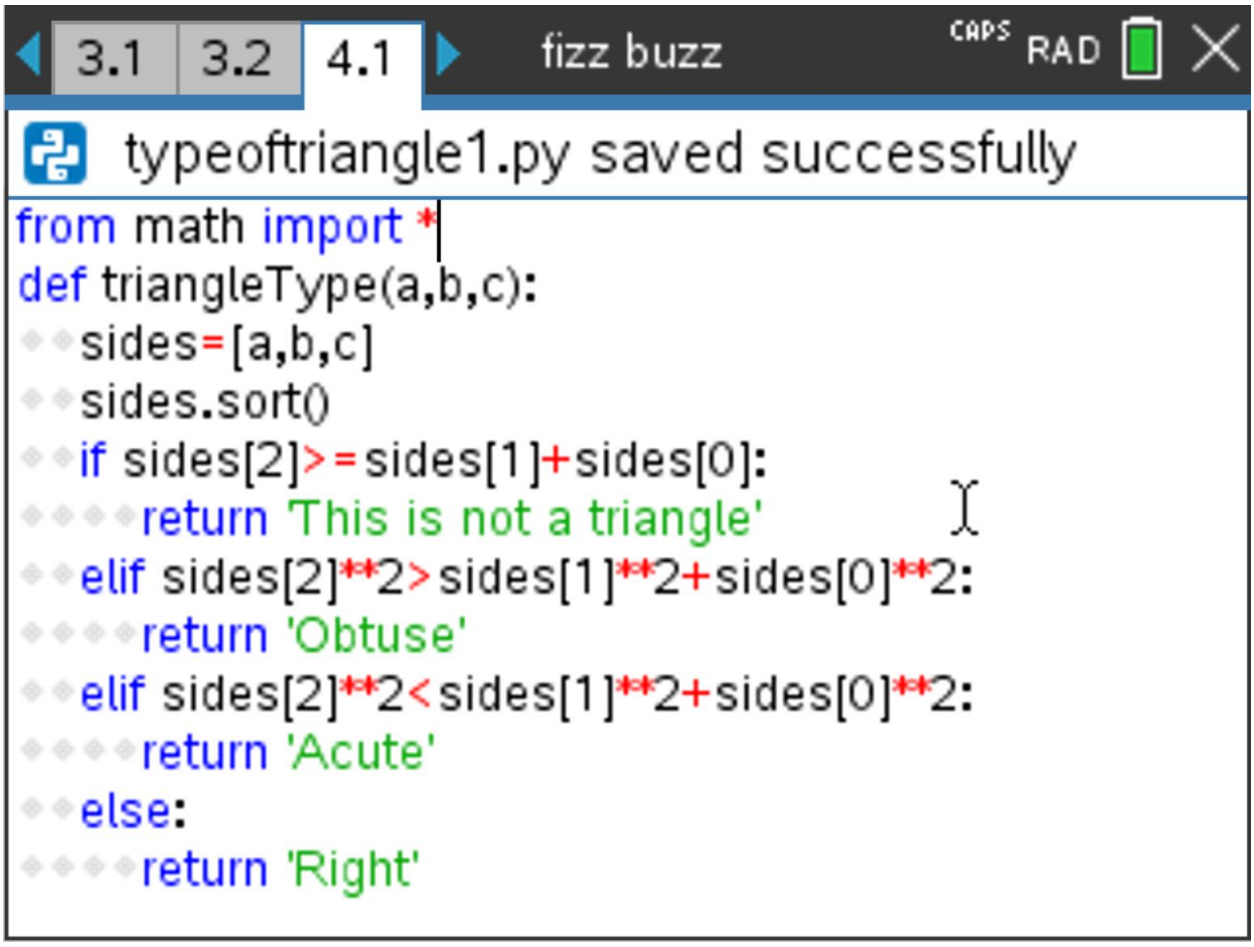
## Other samples from TI-Nspire CX II

## Equation of a Line in Standard Form

Call the function line(x1,y1,x2,y2) which returns the equation of a line that passes through the points (x1,y1) and (x2,y2)

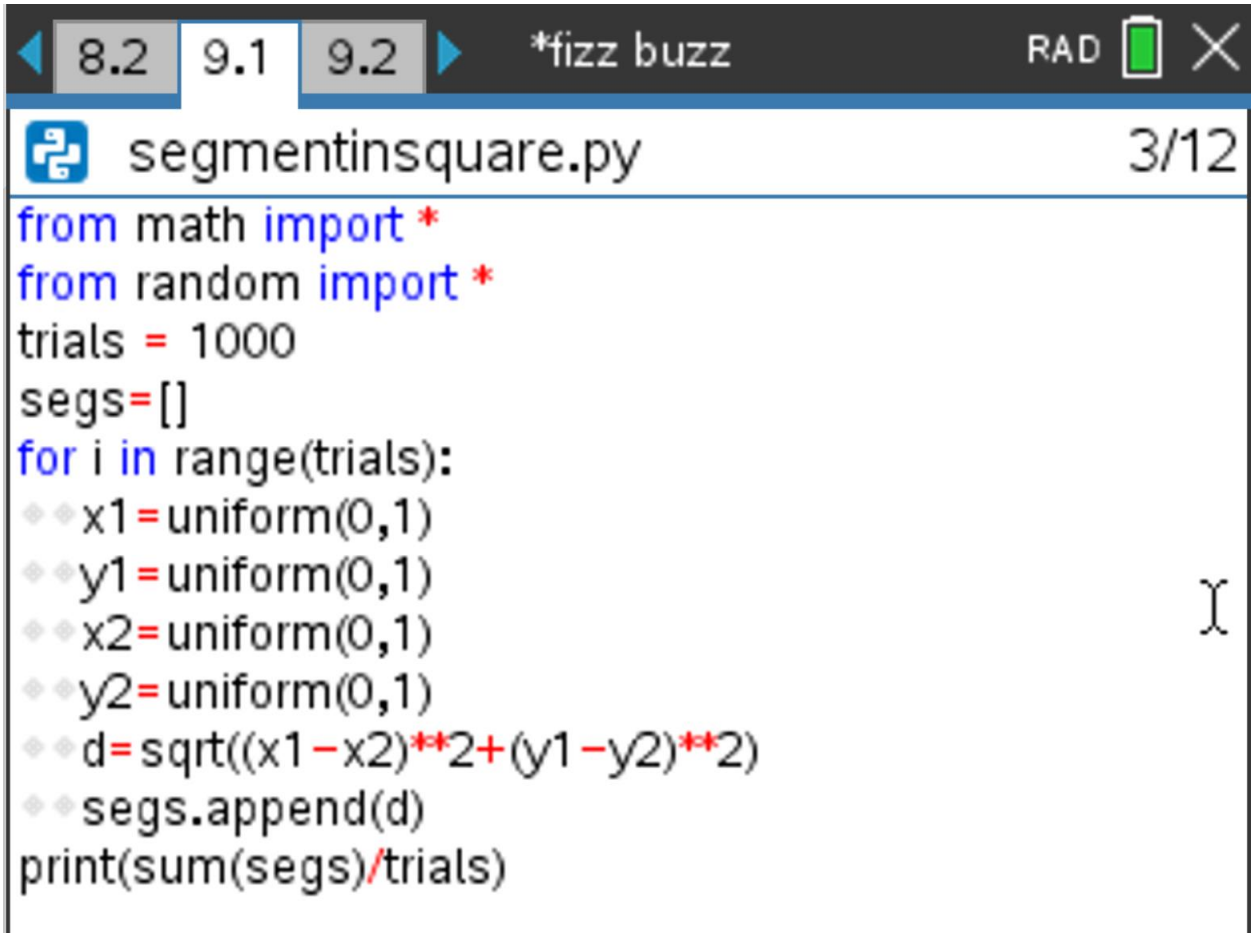


**What Type of Triangle?**



## Segments Inside a Square

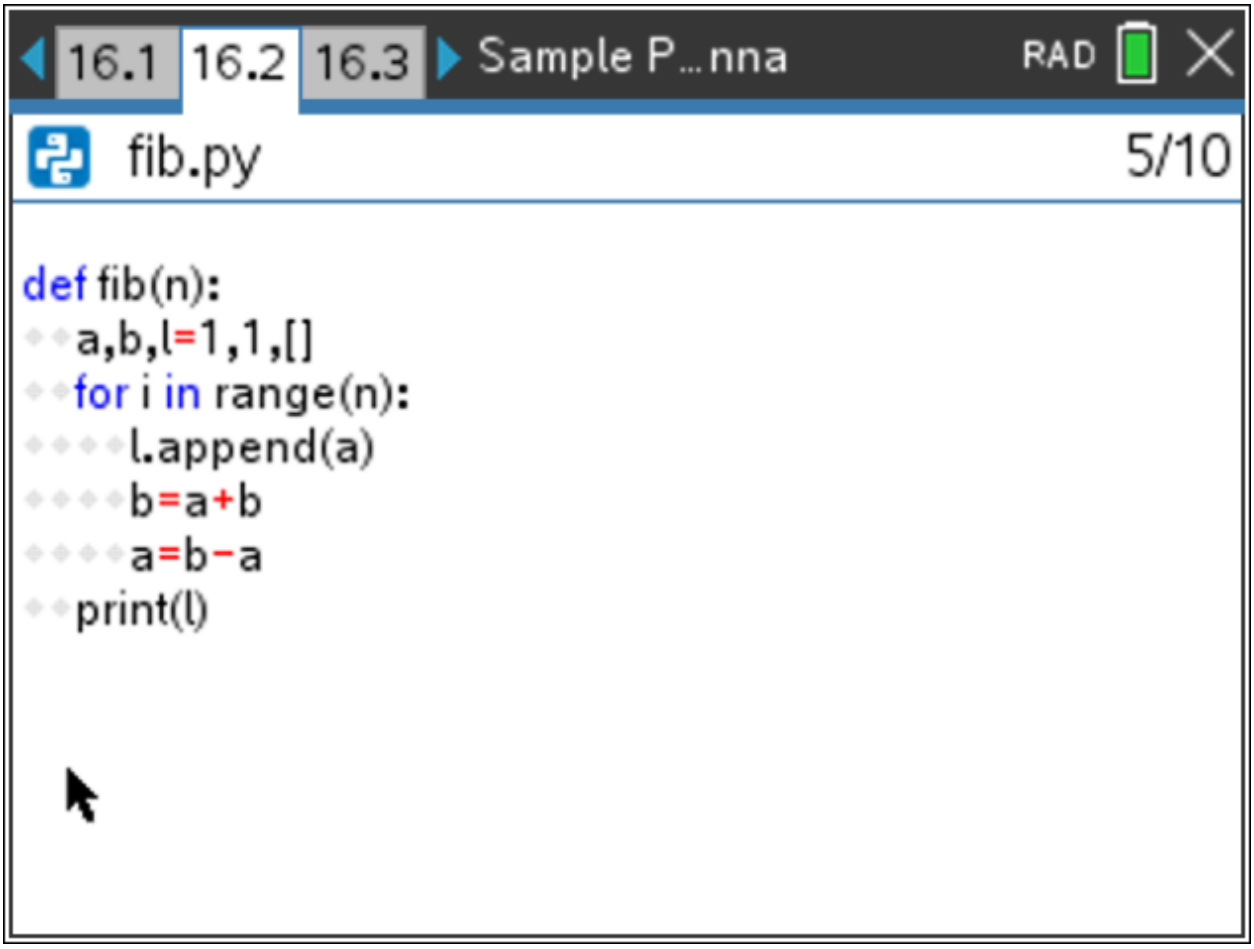
Draw two random points inside a square of side length 1 and connect them with a segment. What is the average length of the typical segment drawn in this manner?



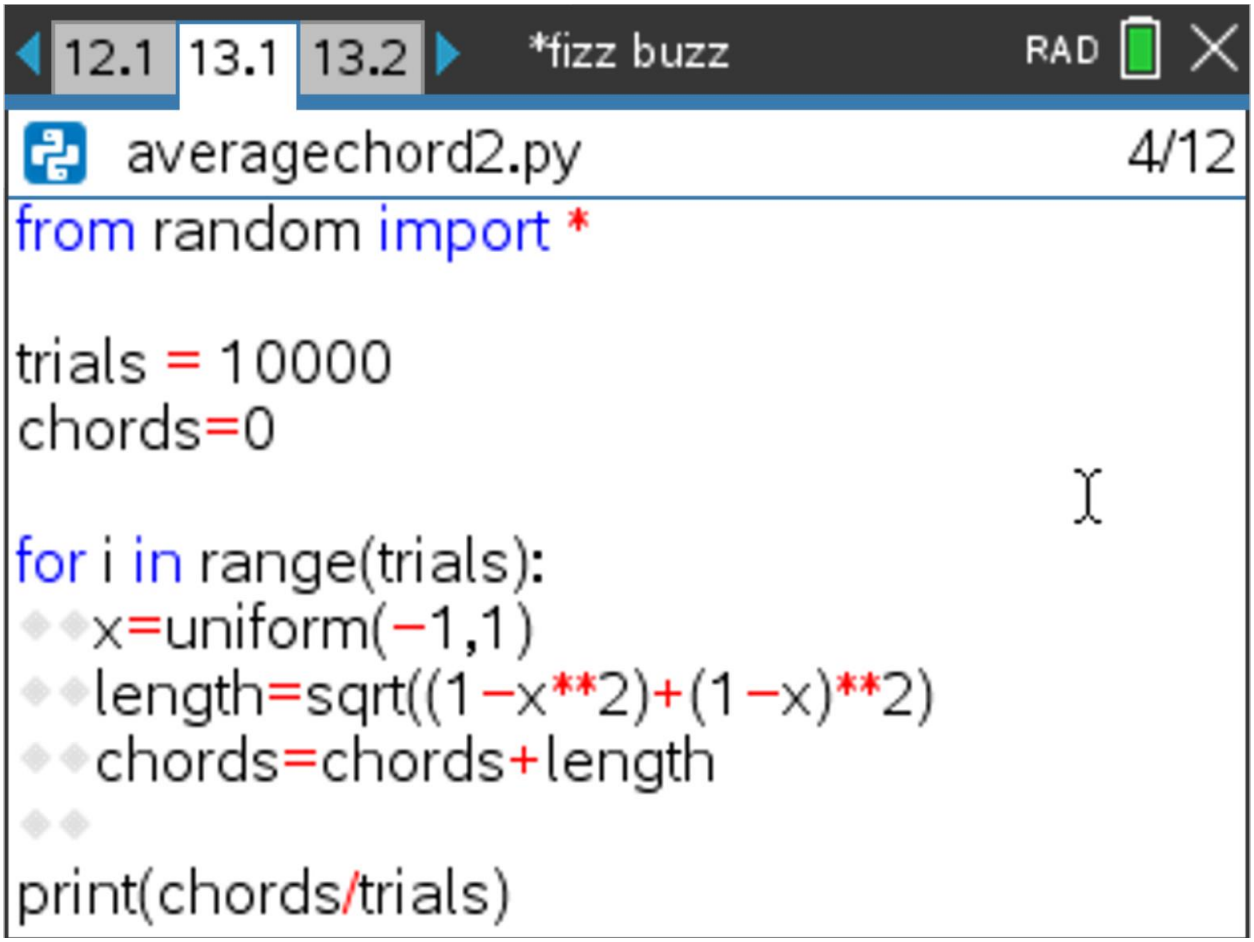
## Fibonacci Sequence

Call the function fib( ) to create a sequence that is formed like the Fibonacci sequence. So…

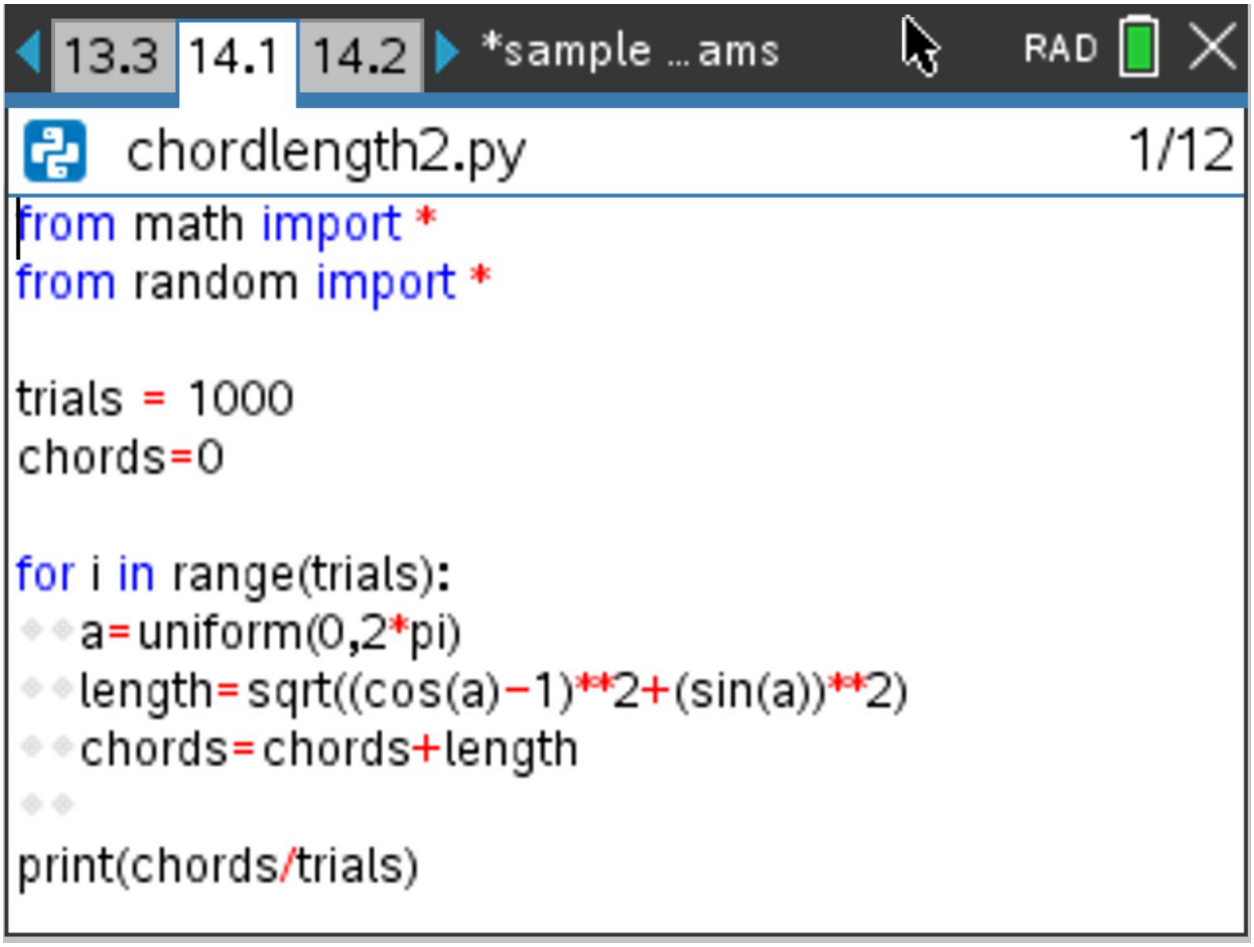
fibo(10) produces [0,1,1,2,3,5,8,13,21,34,55] and so on. It also displays the ratio of the last to numbers in the sequence. the function generates two numbers in each iteration. The print() function prints the requested number of numbers.



**Average Length of a chord (version 1)**

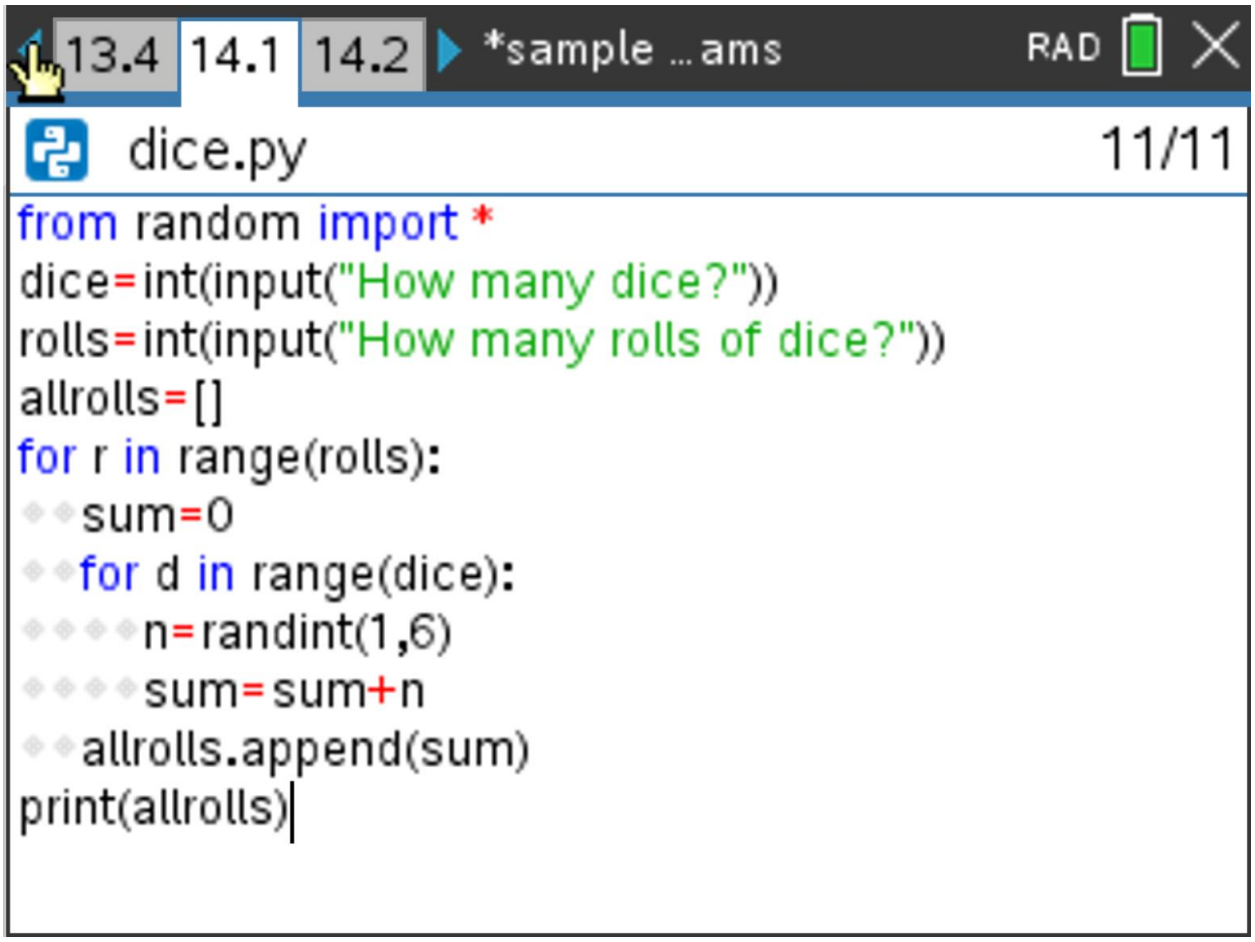


**Average Length of a Chord (version 2)**



## Roll dice and record their sums

Run this program, choose the number of dice to roll, and choose how many rolls (trials) to perform. This will record the sums of all your rolls.



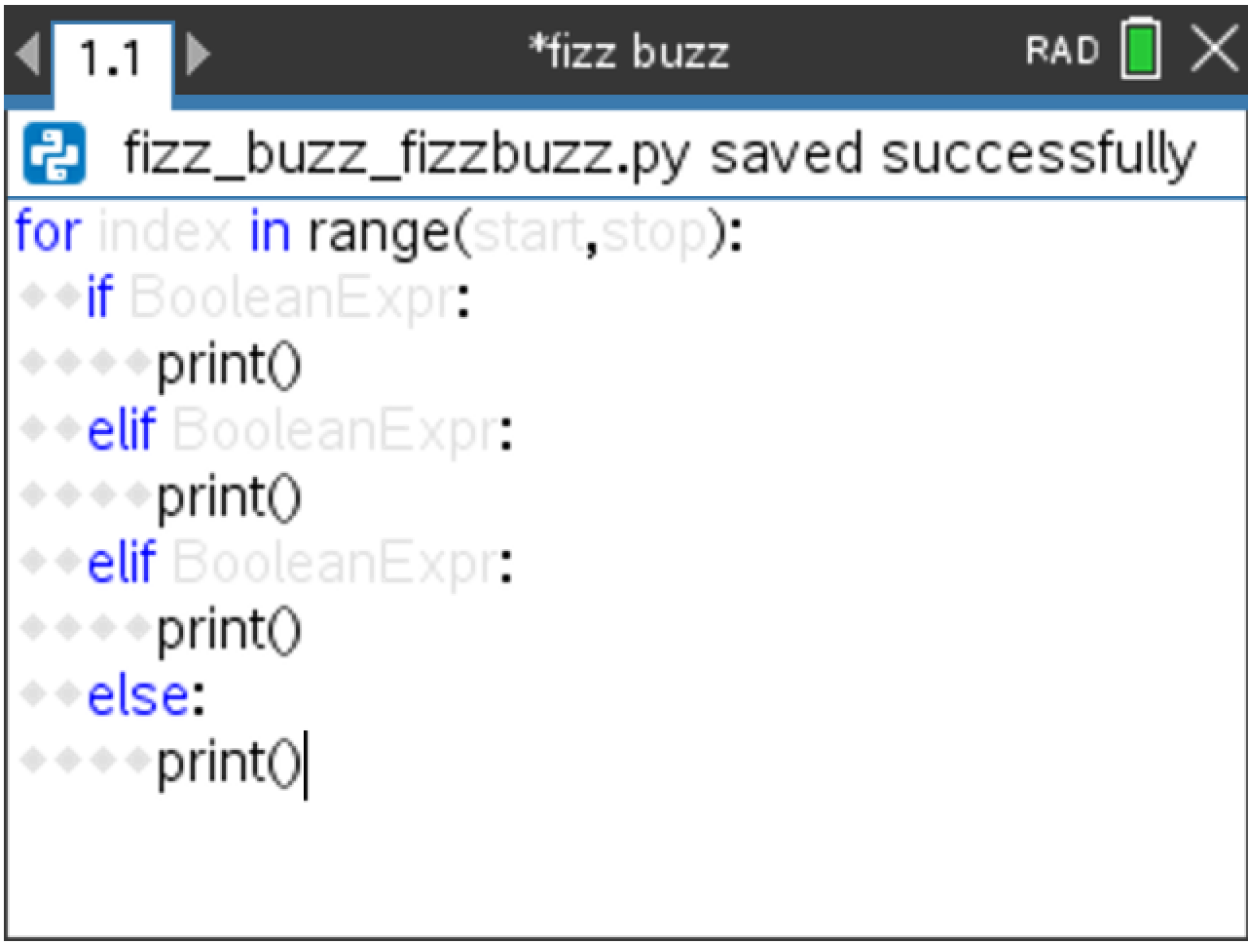
# More Projects

**Project: “Fizz Buzz FizzBuzz”**

Write a program that prints the counting numbers but replaces multiples of 3 with the word “Fizz”, multiples of 5 with the word “Buzz” and multiples of both 3 and 5 with the word “FizzBuzz”.

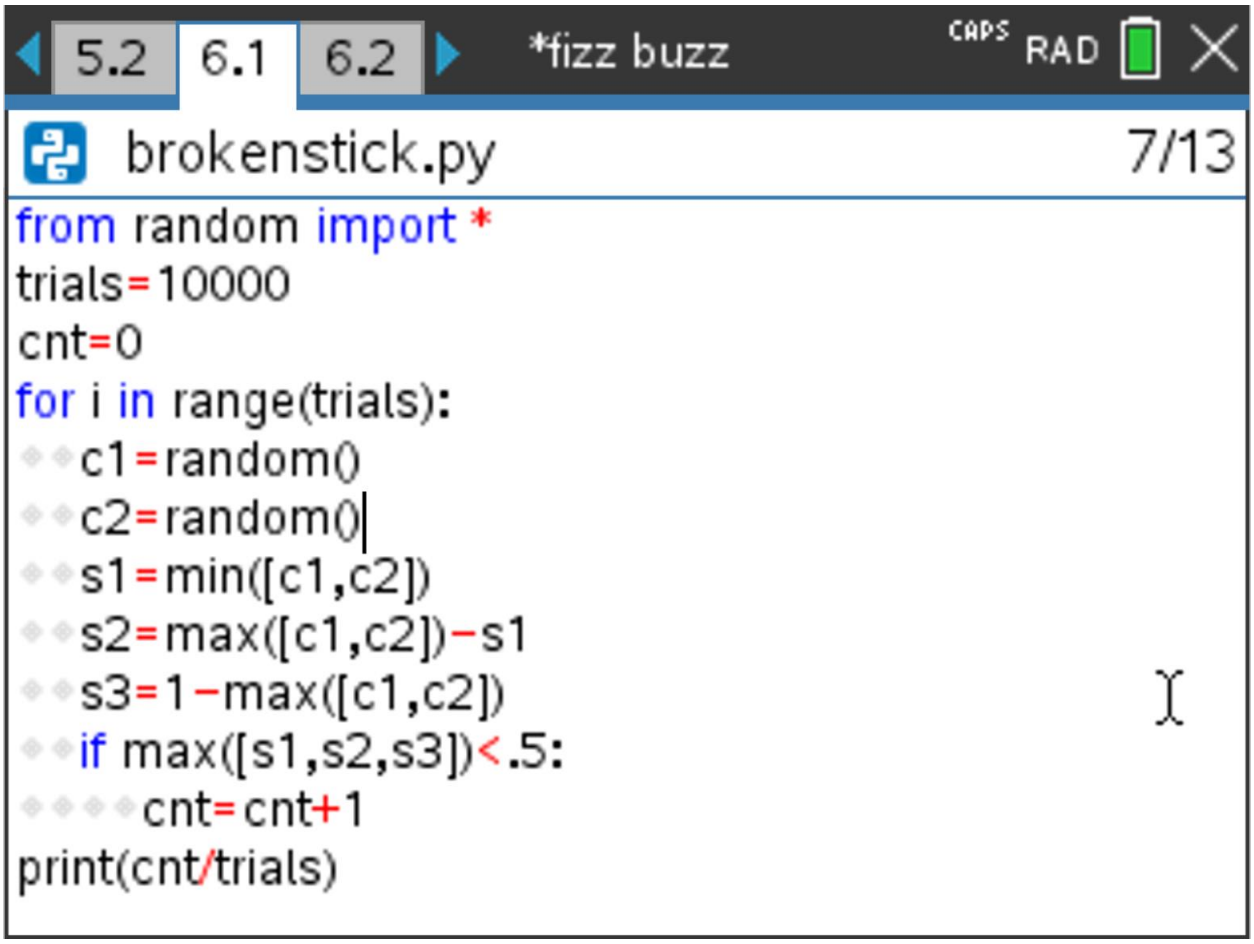
Possible adaptations:

Input a lower and upper limit of numbers



**Broken Stick: Is it a Triangle?**

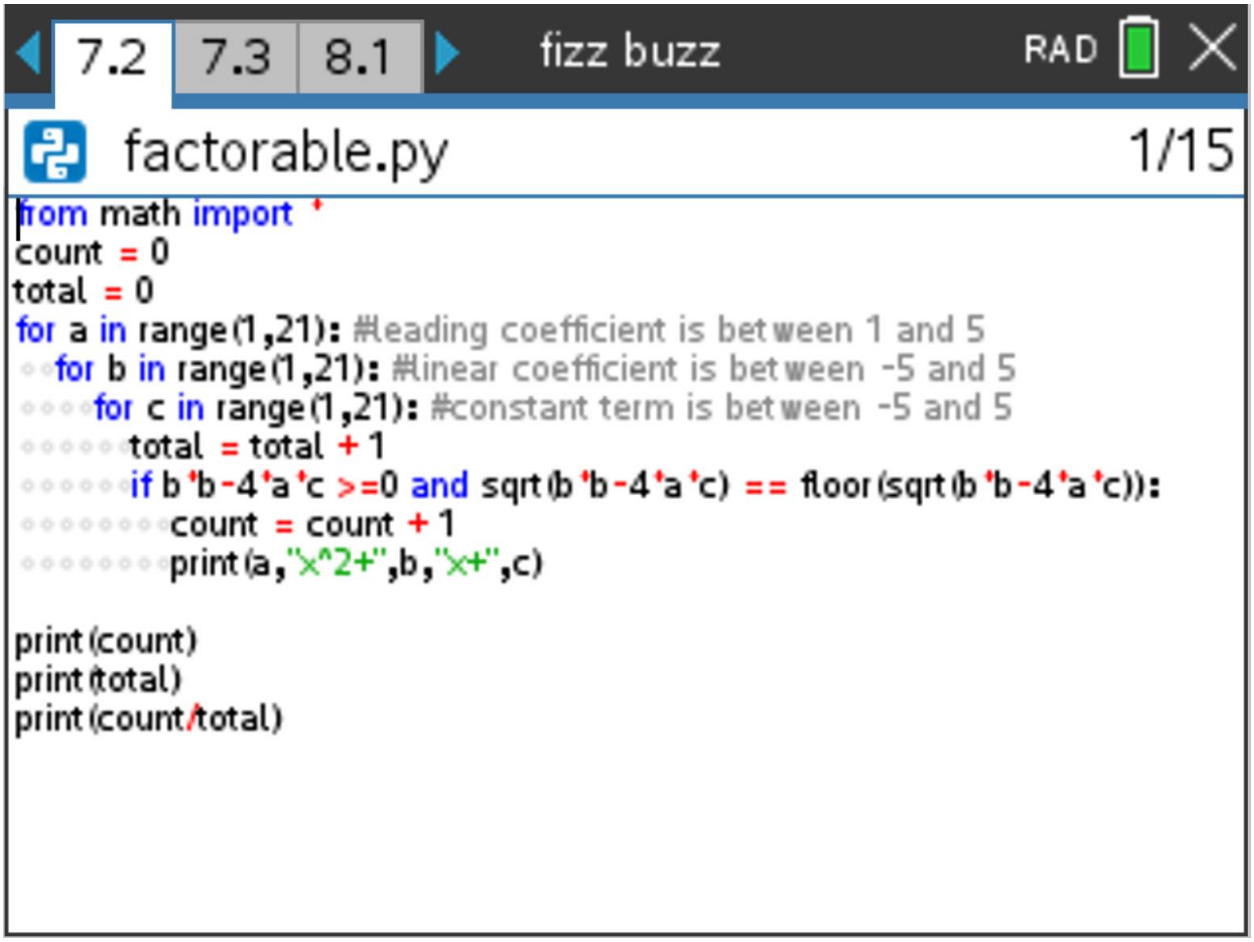
A stick is cut in two random places - c1 and c2 - forming three pieces - s1, s2, and s3 - that are used to form a triangle. Sometimes the pieces form a triangle, sometimes they don't. How likely is it the three pieces form a triangle?



## The Game of Bagels

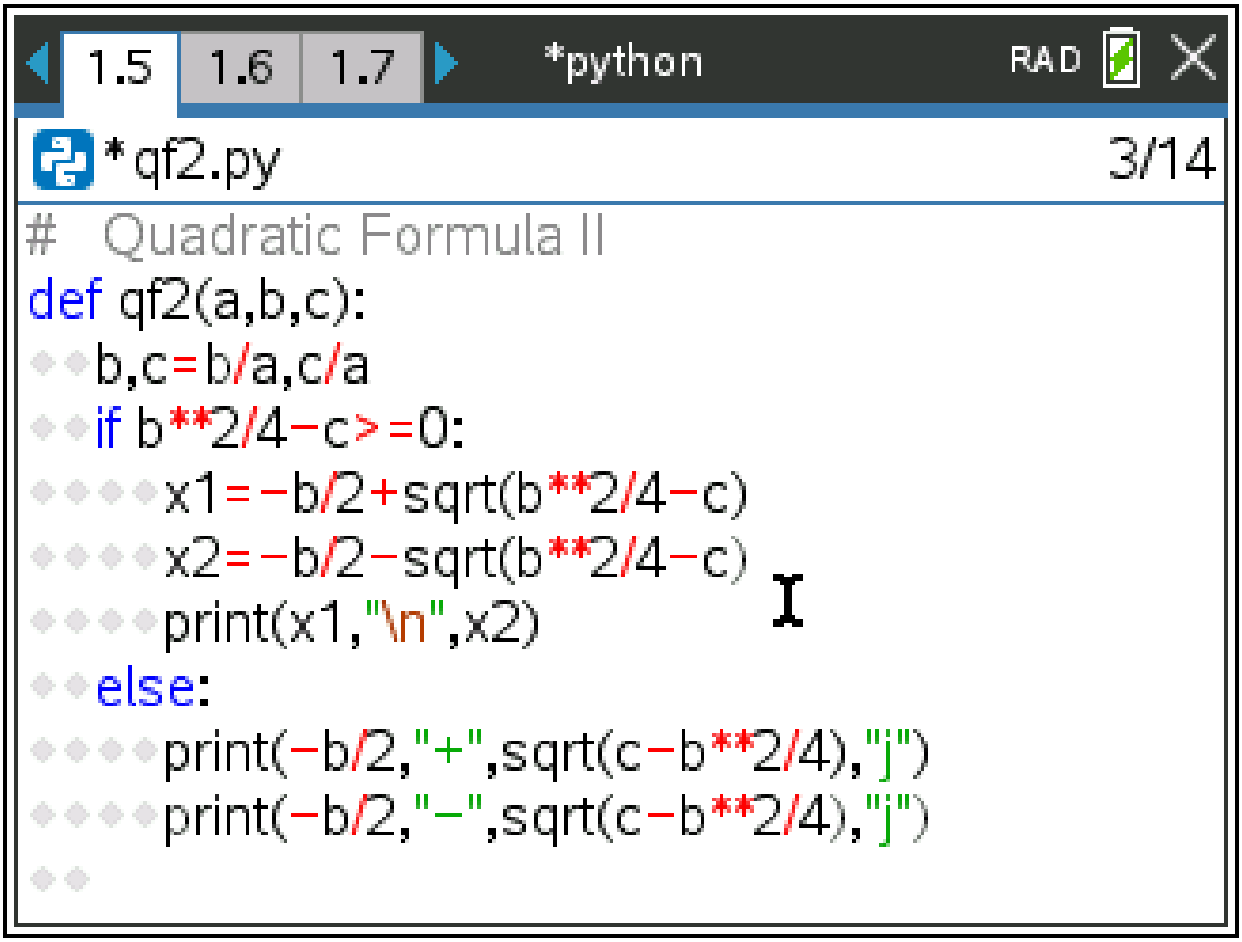
```
# John Hanna, 11/28/2019

from random import *
print("Bagels")
def bagels():
    yn="y"
    while yn=="y":
        count=x=a=b=c=0  #initialize variables
        while a == b or b == c or a == c:
            a = randint(0,9)
            b = randint(0,9)
            c = randint(0,9)
        #endwhile
        ans=(100*a + 10*b + c)
        while x != ans:
            x = int(input("Guess: "))
            if x == 0:print(ans)
            count += 1
            h = int(x/100)
            t = int((x-100*h)/10)
            u = x-100*h-10*t
            str="     "
            if h == a:
                str=str+"Fermi "
            if t == b:
                str=str+"Fermi "
            if u == c:
                str=str+"Fermi "
            if h == b:
                str=str+"Pico "
            if h == c:
                str=str+"Pico "
            if t == a:
                str=str+"Pico "
            if t == c:
                str=str+"Pico "
            if u == a:
                str=str+"Pico "
            if u == b:
                str=str+"Pico "
            if str=="     ":str="     Bagels"
            print(str)
        #endwhile
        print("YOU GOT IT IN", count, "guesses.")
        yn=input("Play again (y/n)?")
#end of bagels()
#next line runs the program right away...
bagels()
```

**Is it factorable? (probability that a quadratic is factorable)**



**Related: Quadratic Formula II**

Alternate Quadratic Formula (complex numbers?):



## Reverse Raffle

```
from random import *
r = input("How many numbers?") #input will be a string
numbers=range(int(r)) #int(r) take input and makes it an integer
nn=len(numbers)
count = 0
print "Start", numbers
while len(numbers)>0:
  count = count + 1
  n = random.randint(0,nn-1)
  if n in numbers:
    numbers.remove(n)
  else:
    numbers.append(n)
  print count,n,sorted(numbers)
```

EXAMPLE

How many numbers? 5

Start [0, 1, 2, 3, 4]

iteration, number chosen, updated list.

| | | |
|---|---|---|
| 1 | 2 | [0, 1, 3, 4] |
| 2 | 3 | [0, 1, 4] |
| 3 | 4 | [0, 1] |
| 4 | 4 | [0, 1, 4] |
| 5 | 2 | [0, 1, 2, 4] |
| 6 | 1 | [0, 2, 4] |
| 7 | 0 | [2, 4] |
| 8 | 1 | [1, 2, 4] |
| 9 | 1 | [2, 4] |
| 10 | 4 | [2] |
| 11 | 3 | [2, 3] |
| 12 | 2 | [3] |
| 13 | 2 | [2, 3] |
| 14 | 4 | [2, 3, 4] |
| 15 | 4 | [2, 3] |
| 16 | 3 | [2] |
| 17 | 2 | [] |

## Project: Make Bricks

We want to make a row of bricks that is **goal** inches long. We have some small bricks (1 inch each) and some big bricks (5 inches each). Return True if it is possible to make the goal by choosing from the given numbers of bricks. This is a little harder than it looks and can be done without any loops.

Example function returns:

```
def make_bricks(small, big, goal):
```

make_bricks(3, 1, 8) → True
make_bricks(3, 1, 9) → False
make_bricks(3, 2, 10) → True

Why Is This Hard At All? #1

- You have 2 big bricks and 2 small bricks
- Can you make a row of 7 inches?
- Can you make a row of 8 inches?
- Notice: need small bricks to hit the length exactly

Why Is This Hard At All? #2

- Suppose you have 2 big bricks and 10 small bricks
- Can you make a row of 16 inches?
- Note: use lots of small bricks instead of a big brick