Let's Get Loopy

30pts 3 = 70% 4 = 80% 5 = 85% 6 = 90% 7 = 93% 8 = 96% 9 = 98% 10 = 100% 11 = 105% 12 = 110% 13 = 115%

Can be done in any order, limited feedback for 1-8 (try examples & use HINTS) - Approach No No's

1. [**01sal**] Mr. Mueller has negotiated a new monthly salary schedule for his teaching job. He will be paid $0.01 on the first day of the month and the daily rate will double each day after. Write a program that will find his daily & total earnings for 30 days. Use the following print statement

```
print(f' {i+1:2d}    {dailySalary:10.2f}  {totalSalary:12.2f}')
```

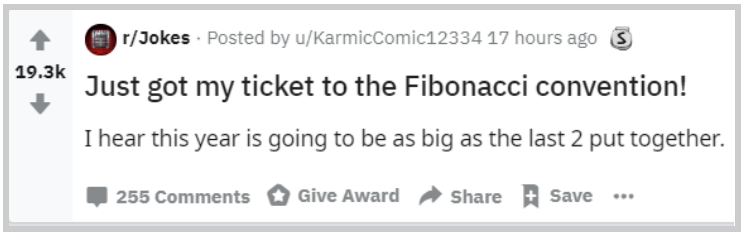
to print each line of the table of your results (don't forget to print the header line first though).

```
Day  Daily Salary  Total Salary
  1          0.01          0.01
  2          0.02          0.03
  3          0.04          0.07
  4          0.08          0.15
  ⋮             ⋮             ⋮
```

2. [**02fib**] The Fibonacci Sequence is the series of numbers

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

Starting with 0 & 1, the next number in the sequence is always found by adding up the two preceding numbers. What is the first Fibonacci number larger than **one billion?**



Hint ***previousNum***, ***currentNum***, ***nextNum*** make great variable names for your fibonaccis.

The first fibonacci number greater than a billion is #. **NOTE: # should be the number.**

3. [**03days**] If you got $1 the first day, $2 the second day, $3 the third day, and so on, after how many days would you have received a total of at least one **billion** dollars? To test your

program you can print the day and the money inside the loop to see if it really takes 14 days to get over $100. **Hint** your program needs to add up the sequence

1 + 2 + 3 + 4 + ...

After # days I would have a billion dollars.

4. [**04agtb**] Country A has **50 million** inhabitants and gains 1000 inhabitants a day. Country B has **100 million** inhabitants and loses 627 inhabitants a day. Assume each year has 365 days (i.e. don't worry about leap years).
Exactly when will country A have more inhabitants than country B?

Country A will have more inhabitants in # years and # days.

5. [**05pi**] The mathematician Gottfried Leibniz determined this formula for estimating the value of PI

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \ldots$$

Evaluate the first 10000 terms of the sequence and print out pi in a formated string using **`{pi:8.7f}`**. For fun - How close to the real value of PI is this value? Is it higher or lower? What happens if you use one more term? two more terms?

HINT For the above sequence of 6 terms you should print **`pi approximation = 2.9760462`**

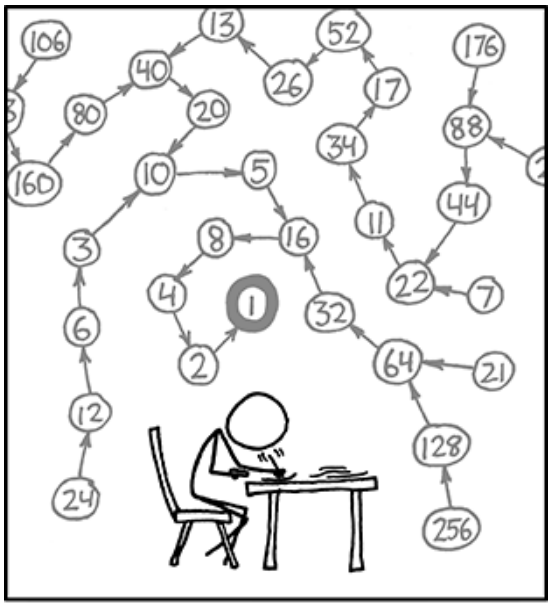
pi approximation = #

6. [**06collatz**] The Collatz conjecture states that for any positive integer repeatedly applying the following procedure

if the number is even, divide it by 2,
else multiply it by 3 and add 1

will always eventually reach the integer value **1**.

Computers have been used to show that the conjecture is correct up to very very large numbers. However, this ***computer evidence*** is not a proof that the conjecture is always true.
Using the initial number **27**, print out how many numbers are in the sequence to get to 1. That is, how many numbers are in the sequence (**27**, 82, 41, 124, **...** , 4, 2, **1**). **Also** print out the largest number in this sequence. NOTE: If you had started with **5** you would have **6 numbers** in the sequence (**5**, 16, 8, 4, 2, **1**) with the largest number in the sequence being **16**.



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

Collatz sequence starting at 27 has # numbers (largest number is #).
NOTE the #s above should both be integers.

7. [**07power**] What is the **only natural number other than 1** that is equal to the sum of its digits raised to a power equal to that digit? In math we would say that the natural number *n* composed of the *k* digits $d_k\ d_{k-1}\ \ldots\ d_2\ d_1$ must be equal to

$$d_k^{\,d_k} + d_{k-1}^{\,d_{k-1}} + \quad \cdots \quad + d_2^{\,d_2} + d_1^{\,d_1}$$

For example, the number **27** (for which d_2 is 2 and d_1 is 7) is NOT the number because $\mathbf{2^2 + 7^7}$ is NOT equal to **27**. Also **451** is NOT the number since $\mathbf{4^4 + 5^5 + 1^1}$ is not equal to **451**. Your program must stop as soon as it has found the number. **HINT** You could convert the number you are currently testing to a string and then access each character (i.e. digit) of this string one character at a time using a for loop - e.g.

```
for digit in "1234":
    print(digit)
```

 # is equal to the sum of its digits raised to a power equal to that digit.

8. [**08triplet**] A Pythagorean triplet is a set of three numbers where **a < b < c** and for which $\mathbf{a^2 + b^2 = c^2}$. For example for the three numbers 3,4, and 5 the following is true $3^2 + 4^2 = 5^2$. There exists exactly one Pythagorean triplet for which **a + b + c = 1000**. Your program should find this triplet in less than 5 seconds.

Hint Write **c** in terms of **a** and **b** using a+b+c=1000 and then use that to write Pythagorean in terms of only **a** and **b** and then try all numbers for 1 to 1000 for both **a** and **b**.

The pythagorean triplet is (a,b,c).

9. [**09polter**] The Poltergeist ride at Six Flags runs every 5 minutes and carries **40** thrill-seekers. Write a function that given
   - the **current** number of people in line
   - the **current** number of new people joining the line every 5 minutes
   - the number of fewer people joining the line every 5 minutes hereafter.

prints
   - how long it takes to reach the longest line length and the number of people in this line
   - how long ***the last person in this longest line*** has to wait (this will be the longest time anyone has to wait in line).

For example, if there are currently **90** people in line, currently **60** new people are joining the line every 5 minutes, and **3** fewer people will join the line every 5 minutes thereafter.

After 5 minutes there will be **90+60-40** = **110** people in line
After 10 minutes there will be **110+57-40** = **127** people in line
After 15 minutes there will be **127+54-40** = **141** people in line

Add this code to call your function twice

poltergeist(90,60,3)
poltergeist(278,68,3)

In # minutes the line will have reached a maximum of # people.
The last person in this line has to wait # minutes.
In # minutes the line will have reached a maximum of # people.
The last person in this line has to wait # minutes.

10. [**10champ**] In mathematics, the ***Champernowne constant*** is a transcendental real constant whose decimal expansion has important properties. It is named after economist and mathematician D. G. Champernowne, who published it as an undergraduate in 1933. The ***Champernowne constant*** is created by concatenating the sequence of positive integers 1,2,3,... immediately following the decimal point:

0.12345678910111213141516171819202 1…

For example, it can be seen that both the **3rd digit (i.e. d_3)** & the **17th digit (i.e. d_{17})** of the fractional part is **3** (both in **red** above).

If $\mathbf{d_n}$ represents the nth digit of the fractional part of the ***Champernowne constant***, find the value of these digits **in less than the allowed time**[1]: d_1 d_{10} d_{100} $d_{1,000}$ $d_{10,000}$ $d_{100,000}$ $d_{1,000,000}$

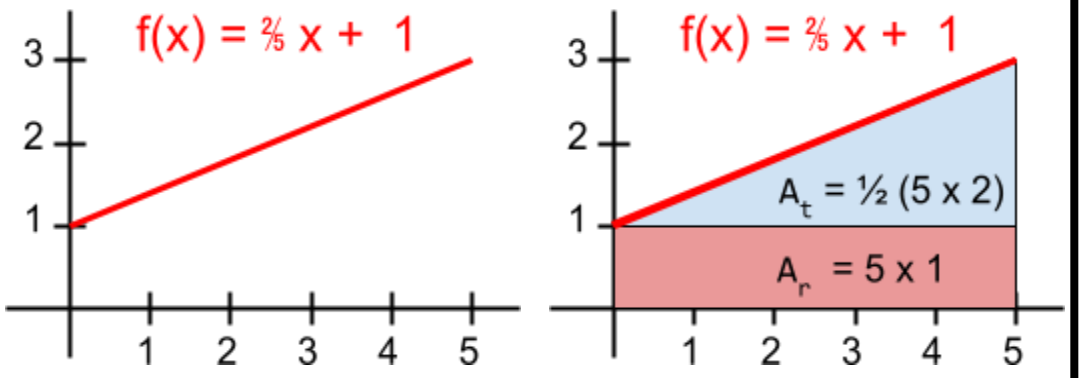
Hint generate a string **`varName`** that contains all the digits and then get each required digit by using **`varName[#]`** to get the character at the position **#** (e.g. if the variable ***myString* = *"abcdef"***, then **`myString[0]`** will get 'a' and **`myString[3]`** will get 'd'). You can use the len() function to tell you the length of the string - e.g. **`print(len(myString))`** would print 6.

[1]**see code in box**
The code on the right will tell you both the maximum time your program is allowed to take and how long your program actually took.

```
import time
startTime = time.time()
endTime   = startTime + 0.25
count = 0
while time.time() < endTime:
   count = count + 1
print("Time allowed for Champernowne =",0.25*1250000/count)
startTime = time.time()
# replace this comment with your Champernowne code
print("Champernowne time =",time.time() - startTime)
```

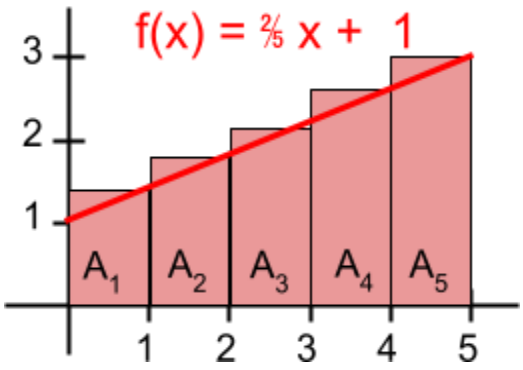
Time allowed for Champernowne = #
Champernowne digits # # # # # # #
Champernowne time = #

11. [11area] The area under a "curve" is the area bounded by a function and the x and y axis. For the function $f(x) = \frac{2}{5}x + 1$ between x=0 and x=5, this area is the sum of the area of the rectangle and the area of the triangle **$A = A_r + A_t = (5 \times 1) + \frac{1}{2}(5 \times 2) =$ 10**



*In Calculus you will learn that the area under **any function** can be **approximated** by dividing the area into vertical rectangles of equal widths and then adding up the area of all the rectangles.*

One way to approximate the area under a curve is to split it into a number of rectangles (each of which may be slightly larger or smaller than what's needed). When splitting the area under $f(x) = \frac{2}{5}x + 1$ **into 5 rectangles** between x=0 and x=5 the total approximate area is given by the area of the 5 rectangles **$A = A_1 + A_2 + A_3 + A_4 + A_5$** where the width of each rectangle is 1 and the height of each rectangle is given by evaluating the function **f(x)** at **x=1,2,3,4,** and **5**.



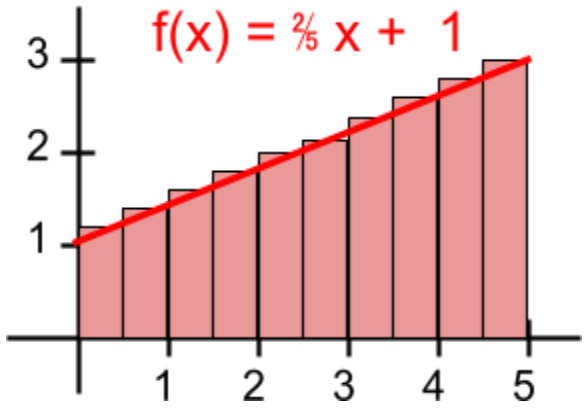
Calculating the total area of the 5 rectangles gives

A = (1 x f(1)) + (1 x f(2)) + (1 x f(3)) + (1 x f(4)) + (1 x f(5))
A = (1 x 7/5) + (1 x 9/5) + (1 x 11/5) + (1 x 13/5) + (1 x 15/5)
A = 55/5
A = 11

which is a little larger than the actual area.

*However, using **more rectangles** gives a better approximation of the actual area.*

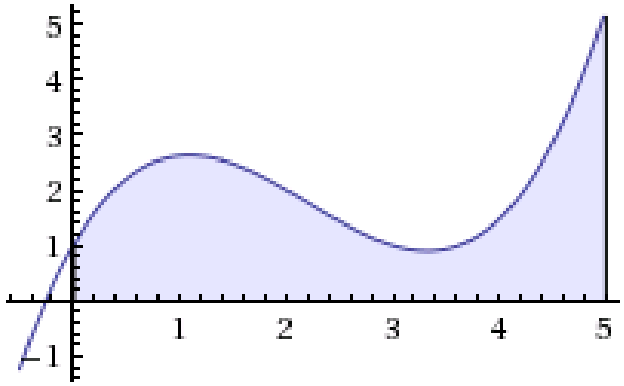
For example, with 10 rectangles the width of each rectangle becomes $\frac{1}{2}$ and a more accurate area is calculated as



A = $\frac{1}{2}$f(0.5) +$\frac{1}{2}$f(1) +$\frac{1}{2}$f(1.5) +$\frac{1}{2}$f(2) +$\frac{1}{2}$f(2.5) +$\frac{1}{2}$f(3) +$\frac{1}{2}$f(3.5) +$\frac{1}{2}$f(4) +$\frac{1}{2}$f(4.5) +$\frac{1}{2}$f(5)
A = 6/10 + 7/10 + 8/10 + 9/10 + 10/10 + 11/10 + 12/10 + 13/10 + 14/10 + 15/10
A = 105/10
A = 10.5

For a given function **f(x)**

- write a **Python function**, that given a value for x **returns** the height of the rectangle (i.e. the value of **f(x)**).
- Use this new function in a **for loop** or **while loop** approximate the area under the curve **between 0 and 5** by dividing the area into equal width rectangles and summing up the area of all the rectangles.
- Test your functions with $f(x) = \frac{2}{5}x + 1$ used in the previous examples for 5 rectangles. Then try it for 10 & 100 rectangles.
- **Finally use the function** $f(x) = \frac{17x^3}{55} - \frac{45x^2}{22} + \frac{369x}{110} + 1$ **graphed below**



- use another **for loop** to calculate the area under the function by using 10,100,1000,10000, 100000, 1000000 rectangles

```
for rectangles in [10,100,1000,10000,100000, 1000000]:
```

FYI, in Calculus you will learn how to calculate this area using the definite integral[1]

$$\int_0^5 \left(\frac{17x^3}{55} - \frac{45x^2}{22} + \frac{369x}{110} + 1\right)dx = 10$$

(2/5)x+1 area with 5 rectangles is #
(2/5)x+1 area with 10 rectangles is #
(2/5)x+1 area with 100 rectangles is #

Cubic equation's area with 10 rectangles is #
Cubic equation's area with 100 rectangles is #
Cubic equation's area with 1000 rectangles is #
Cubic equation's area with 10000 rectangles is #
Cubic equation's area with 100000 rectangles is #
Cubic equation's area with 1000000 rectangles is #

12. [**12flip**] Write a program that simulates flipping a coin 1000 **times** and record how many heads and tails you get. Then **continue** to flip the coin until you *either* have the same number of heads and tails ***or*** until you have 100 more of one then the other. Your program should **repeat** this experiment **10 times** and flip the coin only as much as absolutely necessary. Your output should use the same format as the example below.
Use randint(0,1) to generate a random number (where 0=tails and 1=heads).

After you get your program worked and have tested it, put the statement

```
random.seed(123456)
```

at the beginning of the program (so it executes one time before you start generating random numbers). This will guarantee that the random numbers generated will be the same for you and me and you should then see the exact result below.

```
Experiment #1 first 1000 flips, heads = 497 tails = 503
equal after 30 more flips, heads = 515 tails = 515

Experiment #2 first 1000 flips, heads = 496 tails = 504
equal after 86 more flips, heads = 543 tails = 543

Experiment #3 first 1000 flips, heads = 506 tails = 494
```

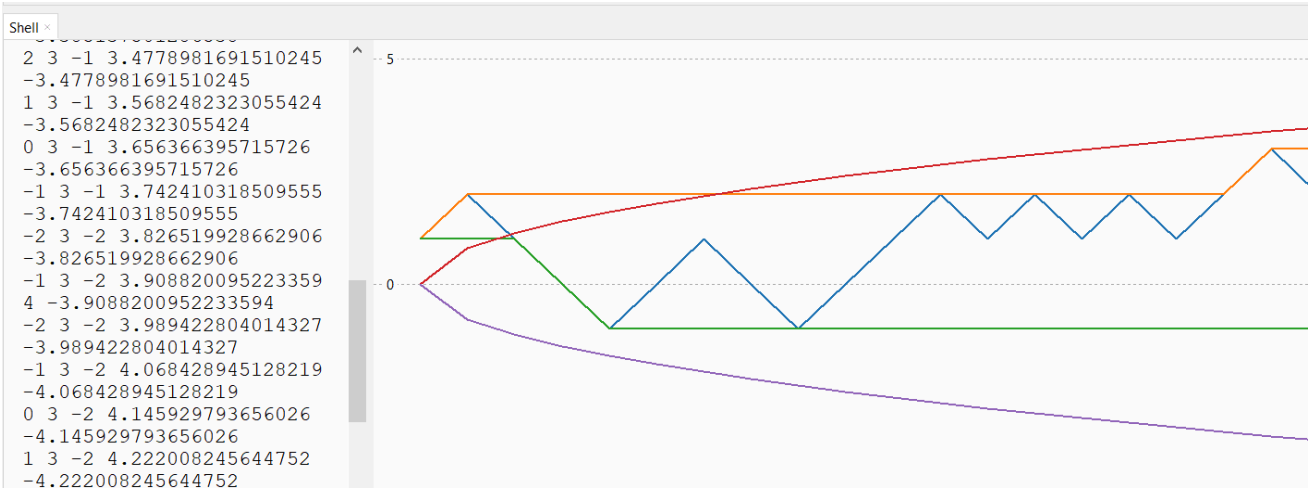
[1] Nov 11, 1675 – Gottfried Leibniz demonstrated integral calculus for the first time to find the area under the graph of good old y=f(x).

```
equal after 642 more flips, heads = 821 tails = 821

Experiment #4 first 1000 flips, heads = 514 tails = 486
equal after 1994 more flips, heads = 1497 tails = 1497

Experiment #5 first 1000 flips, heads = 521 tails = 479
difference is 100 after 2698 more flips, heads = 1899 tails = 1799

Experiment #6 first 1000 flips, heads = 512 tails = 488
equal after 308 more flips, heads = 654 tails = 654

Experiment #7 first 1000 flips, heads = 491 tails = 509
equal after 226 more flips, heads = 613 tails = 613

Experiment #8 first 1000 flips, heads = 511 tails = 489
equal after 490 more flips, heads = 745 tails = 745

Experiment #9 first 1000 flips, heads = 505 tails = 495
equal after 194 more flips, heads = 597 tails = 597

Experiment #10 first 1000 flips, heads = 487 tails = 513
equal after 578 more flips, heads = 789 tails = 789
```

13. **Show it to me in person first then submit it** [**13coin**] Write a program that continually flips a coin and keeps track of the number of heads and tails seen. Print out the following values on a single line separated by a single space
    - **`heads - tails`**
    - the current largest positive value of **`heads - tails`** that has occurred
    - the current largest negative value of **`heads - tails`** that has occurred
    - $\sqrt{(2/\pi)n}$ where *n* is the number of times the coin has been flipped represents the theoretical expected largest positive value of **`heads - tails`**
    - $-\sqrt{(2/\pi)n}$ where *n* is the number of times the coin has been flipped represents the theoretical expected largest negative value of **`heads - tails`**

    Thonny will plot the values you are printing if you select **View -> Plotter**. For the first 200 flips, plot the results at a rate of 10 per second[1], then plot as fast as you can.

The plot should look something like the one below



[1] Use the python sleep method in the time library to slow down your Python program.