EN 001203 Artificial Neural Networks

E001: Brush Up Basic Programming Skills

Faculty of Engineering, Khon Kaen University

Submission: https://autolab.en.kku.ac.th

===

* Submit an answer to a question with a file with `txt` extension. E.g., an answer for Q1 should be submitted in a text file "`Q1.txt`"

* Submit a program to a (programming) problem with a file with a proper extension. E.g., a python program for P2 should be named "`P2.py`"

* All answers and programs must be packaged together in a tar file.

* Each question or problem is worth 60 points. There are 7 problems, thus 420 points are counted as a total score for this exercise.

===

"I don't think of all the misery,

but of the beauty that still remains."

--- Anne Frank

P1. Orientation. Along with finding drinkable water, looking for food, and building a shelter, navigation is one of the basic wilderness survival skills. One of the simplest trick to find a direction from stars is to see which direction the star has moved from its previous position. If we see the star moves up, the direction we are looking is east. If the star moves down, the direction is west. If the star moves left, we are facing north. If the star moves right, we are facing south. If the star rose diagonally to the left, we are facing north-east. And, so on.

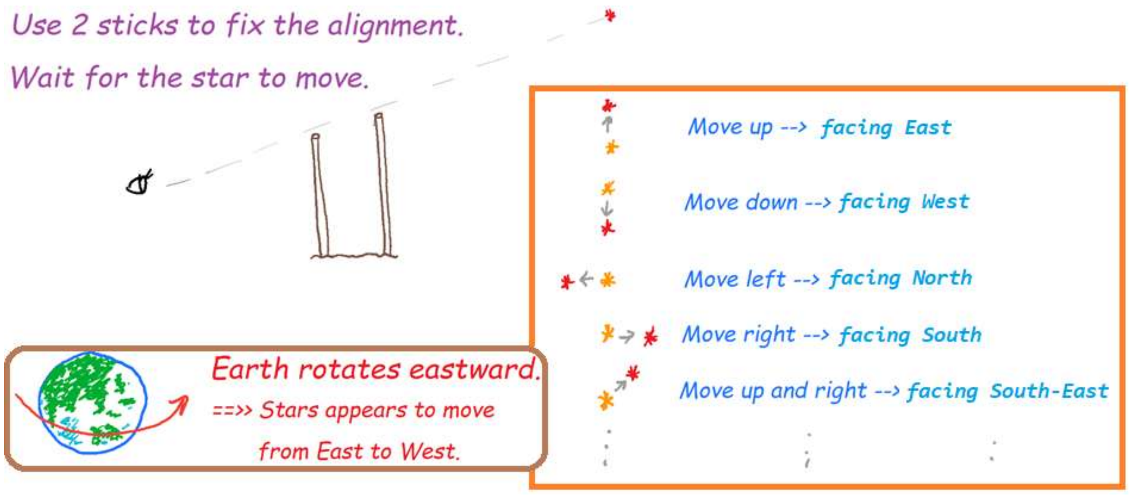

Use 2 sticks to fix the alignment.
Wait for the star to move.
Earth rotates eastward.
==>> Stars appears to move from East to West.
Move up --> facing East
Move down --> facing West
Move left --> facing North
Move right --> facing South
Move up and right --> facing South-East

Figure 1. Orientation by stars.

Although we may not have our computers with us when we need this skill, write a program for it anyway. The program takes an observation code for star moving, figures out the facing direction, and reports the direction.

The observation codes are: '**U**' for star moving up; '**D**' for star moving down; '**L**' for star moving left; '**R**' for star moving right; '**UL**' for star moving up and left; '**UR**' for star moving up and right; '**DL**' for star moving down and left; and '**DR**' for star moving down and right.

Output example, when inputting UL.

```
observe: UL
facing north-east
```

Output example, when inputting U.

```
observe: U
facing east
```

Output example, when inputting R.

```
observe: R
facing south
```

Output example, when inputting DL.

```
observe: DL
facing north-west
```

P2. Online averaging. 0Calculating average value can be re-organized as follows:

$$\overline{x}_N = \frac{x_1+x_2+\cdots+x_N}{N} = \frac{x_1+x_2+\cdots+x_{N-1}+x_N}{N} = \frac{\frac{x_1+x_2+\cdots+x_{N-1}}{N-1}\cdot(N-1)+x_N}{N} = \frac{\bar{\bar{x}}_{N-1}\cdot(N-1)+x_N}{N}.$$

That is, $\overline{x}_1 = x_1$; $\overline{x}_2 = \frac{\overline{x}_1+x_2}{2}$; $\overline{x}_3 = \frac{\overline{x}_2\cdot 2+x_3}{3}$; $\overline{x}_4 = \frac{\overline{x}_3\cdot 3+x_4}{4}$;...

Write a program to keep asking a user for a positive number and compute the average of all numbers having been entered, until the user enters a negative number.

Have the program interact exactly like what shown in the examples. Numbers are supplied by a user.

Output example: when inputting 40, 8, 12, -1.

```
x1: 40
mean=40.0000
x2: 8
mean=24.0000
x3: 12
mean=20.0000
x4: -1
mean=14.7500
```

Output example: when inputting 0.1, 0.2, 0.5, 0, 0.2, -1.

```
x1: 0.1
mean=0.1000
x2: 0.2
mean=0.1500
x3: 0.5
mean=0.2667
x4: 0
mean=0.2000
x5: 0.2
mean=0.2000
x6: -1
mean=0.0000
```

You may use a starter code below.

Starter code:

```
"""
```

```
P2. Online averaging.
Student name: Tenaciti Kurios
"""

if __name__ == "__main__":

    # Write your code here!

    while newx >= 0:

        # Write your code here!

        print("mean={:.4f}".format(xbar))

        # Write your code here!

    # end while
```

P3. EEG. Extract EEG sequence out of a file. Electroencephalography (EEG) is a neuro-observing method to monitor electrical activities of the brain. Using multiple electrodes placed on the scalp, voltages resulting from neural activities of the brain can be sensed. EEG signals are multiple time-series signals. EEG is used in various applications including medical diagnosis, neuroscience research, brain-computer interface, and scientific studies of meditation.

The EEG data used here is taken from:

https://archive.ics.uci.edu/ml/datasets/eeg+database.

The data content looks like

```
# co2a0000364.rd
# 120 trials, 64 chans, 416 samples 368 post_stim samples
# 3.906000 msecs uV
# S1 obj , trial 0
# FP1 chan 0
0 FP1 0 -8.921
0 FP1 1 -8.433
0 FP1 2 -2.574
```

Any line starts with '#' is a metadata (information describing data). Leave the metadata. Extract only sequence data to our variable. The example above has data from line 6 on:

```
0 FP1 0 -8.921

0 FP1 1 -8.433

0 FP1 2 -2.574
```

Details are described in the source website. In short, it is data with multiple time-series signals. Each time-series signal is a sequence of numbers, e.g., `[-8.921, -8.433, -2.574]`. The last column has the time-series values. The third column has their time index. The first and second columns indicate a trial number and a sensor position of the time-series.

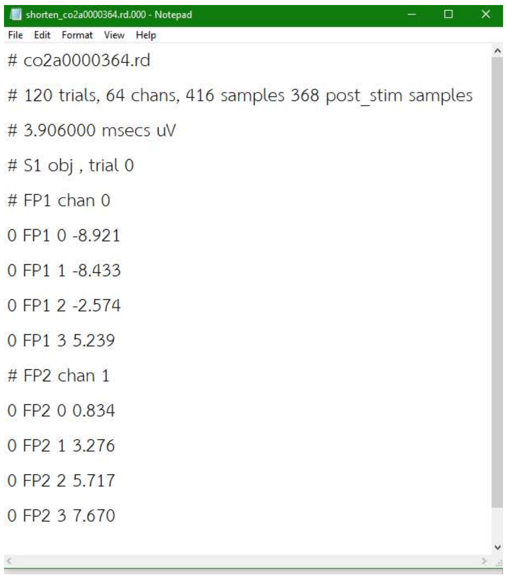
Given this EEG format, write a program to (1) ask a user for EEG file name, trial number, and sensor position, then (2) take a series of the EEG values associating to the specified trial number and sensor position and print it out.

[Hint: it is like navigating the file line by line and for each line take the last part, when the first two parts are matched.]

Have the program interact exactly like what shown in the examples. A file name, trial number, and sensor position are supplied by a user.

Interacting example: when inputting `shorten_co2a0000364.rd.000, 1`, `FP2.`, when the contents of `shorten_co2a0000364.rd.000` is shown in Figure 2.

```
File: shorten_co2a0000364.rd.000
Series: 0 FP2
[0.834, 3.276, 5.717, 7.67]
```


shorten_co2a0000364.rd.000 - Notepad
File Edit Format View Help

```
# co2a0000364.rd
# 120 trials, 64 chans, 416 samples 368 post_stim samples
# 3.906000 msecs uV
# S1 obj , trial 0
# FP1 chan 0
0 FP1 0 -8.921
0 FP1 1 -8.433
0 FP1 2 -2.574
0 FP1 3 5.239
# FP2 chan 1
0 FP2 0 0.834
0 FP2 1 3.276
0 FP2 2 5.717
0 FP2 3 7.670
```

Figure 2. Contents of shorten_co2a0000364.rd.000

P4. Climate model. One of the simplest (yet informative) climate models is an Energy Balance Model (EBM). EBM views the earth as a single body with a single temperature---averaging temperatures of the earth over spatial locations and over times.

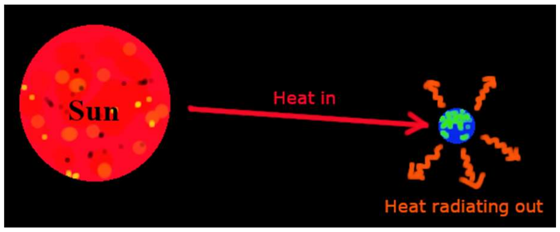


Figure 3. Energy Balance Model

In brief, taking an absorbed heat (1-a)·S and using back body radiation $e\ \sigma\ T^4$, scientists come up with the EBM:

$$e\ \sigma\ T^4 = (1 - a)\ S,$$

where e is emissivity---an amount of energy radiating out into space (no unit); σ is Stefan-Boltzmann constant (5.67×10^{-8} W·m^{-2}·K^{-4}); T is the earth mean temperature (in °K); a is Albedo---accounting for earth reflection of Sun rays by atmosphere and clouds ($a \approx 0.3$); and S is heat from the Sun (in W·m^{-2}).

To have a better intuition, Svante Arrhenius has re-arranged the EBM to:

$$T = \sqrt[4]{\frac{(1-a)\cdot S}{e\sigma}}.$$

This rearrangement allows direct inference on earth mean temperature given major factors influencing it.

The current climate has: solar heat $S = 342$ W·m^{-2} and albedo $a = 0.31$. The emissivity e depends what in the atmosphere, e.g., amounts of methane, CO_2, and water vapor. This is the pivotal point on “climate change” or “global warming”, a higher level of CO_2 leads to a lower emissivity. Recall from the EBM that a lower e value leads to a higher T. Thus, in short, higher $CO_2 \rightarrow$ lower $e \rightarrow$ higher T. The current value of e is 0.605, but as a level of atmospheric CO_2 is increasing, e is decreasing. That is raising an alarm on global scale disasters, e.g., extreme weathers, crop failures, and rising sea level.

(To learn more, see Chris Budd's talk: The Math of Climate Change, at Museum of London on 13 November 2018, https://www.gresham.ac.uk/lectures-and-events/mathematics-climate-change.)

Write a program to compute the EBM taking solar heat S, albedo a, and emissivity e and report the earth mean temperature T in Kelvin. Use $\sigma = 5.67 \times 10^{-8}$ W·m^{-2}·K^{-4}.

[Hint: how many ways can we do the fourth root?]

Have the program interact exactly like what shown in the examples. Solar heat, albedo, and emissivity are supplied by a user.

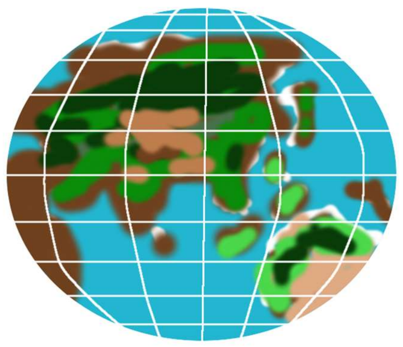
Output example: when inputting 342, 0.31, 0.605.

```
S: 342
a: 0.31
e: 0.605
T= 287.99
```

Output example: when inputting 342, 0.31, 0.596.

```
S: 342
a: 0.31
e: 0.596
T= 289.08
```

P5. Old-time sailor. Latitude and longitude are units as coordinates to pinpoint a location on earth surface. Latitude is described by imaginary horizontal lines. The 0-degree line is at the equator. The other lines lie parallel to the equator. The North Pole is at 90 degree north. The South Pole is at 90 degree south. Longitude is described by imaginary vertical lines running from the North Pole to the South Pole. The 0-degree line passes through Greenwich, England. East/West is to specify whether the line located east or west of Greenwich.



Before the advent of GPS technology, sailors must learn skills to locate their whereabouts. For example, latitude can be deduced from the angle from the horizontal level to the Polaris when they are in the northern hemisphere (Figure 1), or to the South Star (Sigma Octantis) when they are in the southern hemisphere.

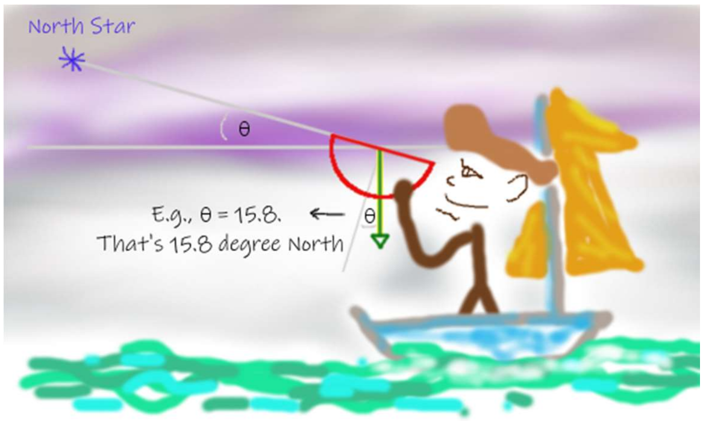
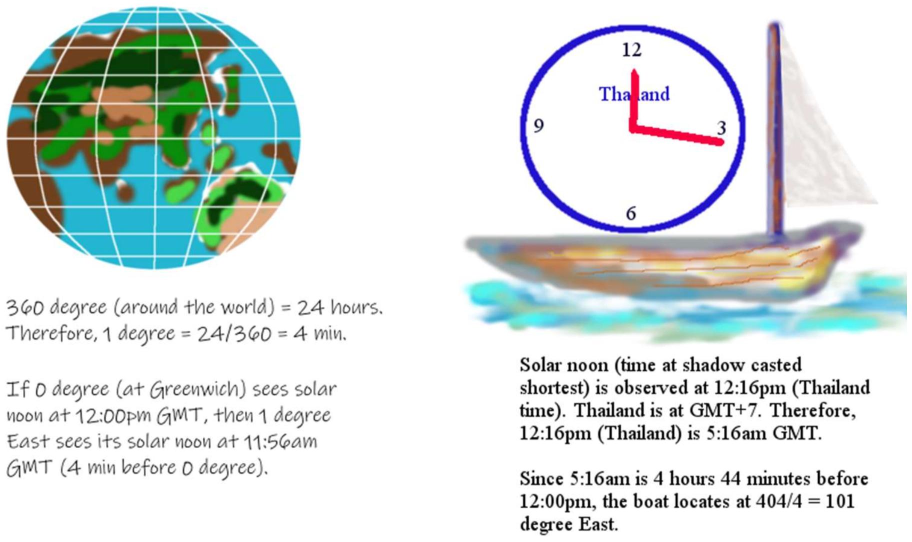


Figure 4. Sailer uses a sectant for determining latitude.

One way to deduce longitude is based on time at solar noon. The solar noon is the time when the shadow appears shortest. Time at solar noon will be compared to 12:00pm GMT, which is the time at solar noon in Greenwich. Since around the world counts 24 hours, hence 4-minute difference corresponds to 1 degree longitude. (Total 360 degrees around the globe counts 1440 minutes, which are 24 hours.) If GMT at solar noon in the location is before 12:00pm, the location lies east of Greenwich. Otherwise, the location is lies west.



360 degree (around the world) = 24 hours.
Therefore, 1 degree = 24/360 = 4 min.

If 0 degree (at Greenwich) sees solar noon at 12:00pm GMT, then 1 degree East sees its solar noon at 11:56am GMT (4 min before 0 degree).

Solar noon (time at shadow casted shortest) is observed at 12:16pm (Thailand time). Thailand is at GMT+7. Therefore, 12:16pm (Thailand) is 5:16am GMT.

Since 5:16am is 4 hours 44 minutes before 12:00pm, the boat locates at 404/4 = 101 degree East.

*Figure 5. How longitude can be determined from GMT. Solar noon at 5:16am GMT = 5*60 + 16 = 316 minutes. Since 12:00pm = 720 minutes, thus 5:16am is 316 - 720 = -404 minutes or 404 minutes before 12:00pm.*

Write a program to take a GMT at solar noon and report the longitude.

[Hint: convert GMT to a number of minutes from midnight; then $longitude = \left|\frac{GMT\ (in\ minutes) - noon\ (in\ minutes)}{4.0}\right|$ and if GMT < noon, it is E; if GMT > noon, it is W.]

Have the program interact exactly like what shown in the examples. A GMT is supplied by a user.

Output example: when inputting 5:16a.

```
GMT: 5:16a
longitude= 101.00 E
```

Note: Thailand locates at around 15.8700° N, 100.9925° E.

Output example: when inputting 7:03p.

```
GMT: 7:03p
longitude= 105.75 W
```

Output example: when inputting 11:40p.

```
GMT: 11:40p
longitude= 175.00 W
```

Output example: when inputting 12:00p. (Exact longitude of Greenwich.)

```
GMT: 12:00p
longitude= 0.00
```

Output example: when inputting 12:00a.

```
GMT: 12:00a
longitude= 180.00 E
```

You may use a starter code below.

Starter code:

```
"""
P5. Old-time sailor.
Student name: Tenaciti Kurios
"""

def time2min(timetxt):

    meridiem = timetxt[-1]
    h, m = timetxt[:-1].split(":")

    mins = 0
    if meridiem == 'p':
        mins = 720 # offset the pm time to 720 mins
    # end if
    mins += 60* (int(h)%12) + int(m)    # 12:00a = 0:00a

    return mins


if __name__ == "__main__":
    GMTnoon = 720 # noon = 12:00pm = 12*60 = 720 min

    GMT = input("GMT: ")
    mins = time2min(GMT)

    # Write your code here!

    print("longitude= {:.2f} {}".format(longitude, direction))
# end __main__
```

P6. Base-3 stone code. Uncle A really enjoys swimming. He needs to count a number of lapses he has done using a stone pattern on a pool side. He decides to use a base-3 system, e.g.,

(base-3 code) = decimal			
$(000)_3 = 0$	$(020)_3 = 6$	$(110)_3 = 12$	$(200)_3 = 18$
$(001)_3 = 1$	$(021)_3 = 7$	$(111)_3 = 13$	$(201)_3 = 19$
$(002)_3 = 2$	$(022)_3 = 8$	$(112)_3 = 14$	$(202)_3 = 20$
$(010)_3 = 3$	$(100)_3 = 9$	$(120)_3 = 15$	… so on.
$(011)_3 = 4$	$(101)_3 = 10$	$(121)_3 = 16$	(Uncle A rarely goes over 20. He's tired.)
$(012)_3 = 5$	$(102)_3 = 11$	$(122)_3 = 17$	

Code value: 0_3 means putting a stone on a pool edge; 1_3 means putting a stone in the middle of the pool side; and 2_3 means putting a stone on the furthest edge. See Figure 6 along.

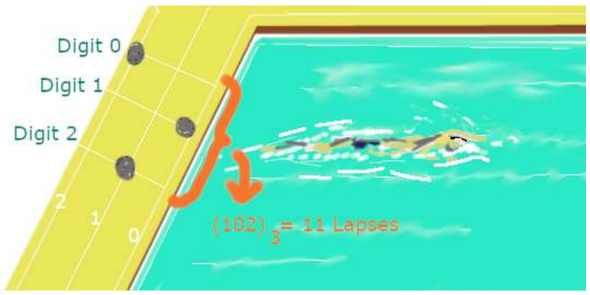


Figure 6. Stone Lapse count using base 3 system

Write a program to take a stone code (a base-3 number) and compute how many lapses (a decimal value of the base-3 number). Uncle A uses only 3 stones, i.e., a stone code has 3 digits. (Do not worry about accommodating a general base-3 number.)

[Hint: $(abc)_3 = 3^2 \cdot a + 3^1 \cdot b + 3^0 \cdot c = 9\,a + 3\,b + c$, e.g., $(111)_3 = 9 + 3 + 1 = 13$; $(202)_3 = 9(2) + 0 + 2 = 20$.]

Have the program interact exactly like what shown in the examples. A stone code (base-3 number) is entered by a user.

Output example: when inputting 201.

```
Stones: 201
Lapses= 19
```

Output example: when inputting 001.

```
Stones: 001
Lapses= 1
```

Output example: when inputting 011.

```
Stones: 011
Lapses= 4
```

P7. SIR model. The SIR model is a mathematical model to estimate the spread of an infectious disease. The SIR model categorizes population into 3 groups: susceptible, infectious, and removed groups. People can be moved between groups.

The susceptible group is a subpopulation of susceptible individuals. Variable ***S*** represents a number of susceptible individuals. Susceptible individuals do not have the disease, but can be infected. When a susceptible individual come into (infectious) contact with an infectious individual, the susceptible individual is infected and re-assigned to the infectious group.

The infectious group refers to individuals who have been infected and can infect susceptible individuals. Variable ***I*** represents a number of infectious individuals. After a period of time, an infectious individual will either recover with immunity or die. Either case, the individual will be moved to the removed group. Note that, a conventional SIR model assumes that a proportion of having been recovered without immunity and become susceptible again is negligible.

The removed group refers to individuals who cannot be infected anymore, i.e., having recovered from the disease with immunity or died. Variable ***R*** represents a number of immuned or deceased individuals. A removed individual will stay in this group: he/she cannot be re-infected nor infect others.

To simplify the matter, an SIR model assumes that: (1) the epidemic is short; the total population remains constant; (2) a rate of increase in ***I*** is constant; and (3) a rate of increase in ***R*** is constant.

That is, the progression of group sizes can be illustrated in Figure 5.

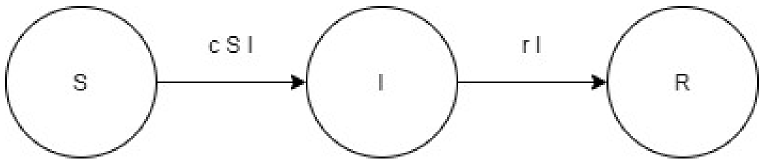


Figure 7. Progression in an SIR model

The progression of the disease is modeled as:

The change in ***S***:

$$\frac{dS}{dt} = -c\,S\,I$$

A number of susceptible individuals is decreased by infection, which depends on the infectious contact. The contact rate ***c*** is assumed constant.

The change in ***I***:

$$\frac{dI}{dt} = c\,S\,I - r\,I$$

A number of infectious individuals is changed by newly infected individuals and newly removed individuals. The removal rate ***r*** is assumed constant.

The change in ***R***:

$$\frac{dR}{dt} = r\,I$$

A number of removed individuals is increased by individuals recovered or died.

To determine whether the disease will spread or be contained, it can be seen from

$$\frac{dI}{dt} = c\,S_o\,I - r\,I,$$

where $\boldsymbol{S_o}$ is an initial susceptible number. If $c\ \boldsymbol{S_o} > r$ (making the change in I positive), the disease will spread. The rates *c* and *r* depend on social practice, countermeasure, and healthcare capability for prevention and intervention. They both can be determined from data through model fitting. (Learn more at *Wikipedia: Compartmental models in epidemiology*.)

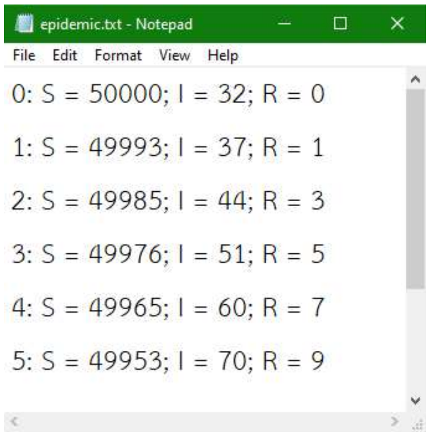
Write a program to take in initial conditions (***S***, ***I***, ***R***), model parameters (*c* and *r*), and a number of periods, then compute the progression of the disease and write the results into a file named `epidemic.txt`, along with summarize if the disease is spreading.

[Hint: the original SIR model is a differential model; here we simplify it to a discrete model.]

Have the program interact exactly like what shown in the examples. Initial conditions, model parameters, and a number of simulated periods are supplied by a user.

Interacting example 1: when inputting 50000, 32, 0, 4.2e-6, 0.04, and 5. Note: 4.2e-6 is a notation for 4.2×10^{-6}. The program also writes `epidemic.txt` file, whose contents are shown in Figure 6.

```
Initial S, I, R: 50000 32 0
c, r: 4.2e-6 0.04
periods: 5
--> It is spreading!
```



```
0: S = 50000; I = 32; R = 0
1: S = 49993; I = 37; R = 1
2: S = 49985; I = 44; R = 3
3: S = 49976; I = 51; R = 5
4: S = 49965; I = 60; R = 7
5: S = 49953; I = 70; R = 9
```

Figure 8. Contents of epidemic.txt afer running P6 (example 1)

Interacting example 2: when inputting 8000, 100, 0, 5e-6, 0.04, and 3. (Contents of the output file is not shown.)

```
Initial S, I, R: 8000 100 0
c, r: 5e-6 0.04
periods: 3
--> It is contained!
```

You may use a starter code below.

Starter code:

```
"""
P7. SIR model.
Student name: Tenaciti Kurios
"""

if __name__ == "__main__":
    S, I, R = input("Initial S, I, R: ").split()
    c, r = input("c, r: ").split()
    S = float(S)
    I = float(I)
    R = float(R)
    c = float(c)
    r = float(r)
    periods = int(input("periods: "))


    # Write your code here!
        print("--> It is spreading!")
    # Write your code here!
        print("--> It is contained!")

    with open("epidemic.txt", "w") as f:
        m =  "0: S = {:.0f}; I = {:.0f}; R = {:.0f}\n".format(S, I, R)
        f.write(m)

        for n in range(periods):
            # Write your code here!

            m =  "{}: S = {:.0f}; I = {:.0f}; R = {:.0f}\n".format(n+1, S, I, R)
            f.write(m)

        # end for
    # end with
```