Your name here.
Your Woskshop section here.

Homework 1: Python basics and a little plotting

** Submit this notebook to bCourses to receive a credit for this assignment.**

Please complete this homework assignment in code cells in the iPython notebook. Include comments in your code when necessary. Enter your name in the cell at the top of the notebook, and rename the notebook [email_name]_HW01.ipynb, where [email_name] is the part of your UCB email address that precedes "@berkeley.edu". Please also submit a PDF of the jupyter notebook to bcourses.

Problem 1: Satellite Altitudes

[Adapted from Newman, Exercise 2.2] A satellite is to be launched into a circular orbit around the Earth so that it orbits the planet once every T seconds. The altitude h above the Earth's surface that the satellite must have is

$$h = \left(\frac{GMT^2}{4\pi^2}\right)^{1/3} - R,$$

where $G = 6.67 \times 10^{-11}\ \mathrm{m}^3\ \mathrm{kg}^{-1}\ \mathrm{s}^{-2}$ is Newton's gravitational constant, $M = 5.97 \times 10^{24}\ \mathrm{kg}$ is the mass of the Earth, and $R = 6371\ \mathrm{km}$ is its radius.

1a. Write a program that, for a given value of T (entered as a variable T in a cell), calculates and prints out the correct altitude in meters, kilometers, and miles, with one decimal place for each result.

Output for 1a: When the code cell for this part of the problem is entered, it should specify (in the comments or the Markdown cell above) what units of T are assumed. The program should print out the correct altitude in meters, kilometers, and miles, with one decimal place for each result.

In [1]:
```
import numpy as np
PI = np.pi  #the numpy library has a lot of useful mathematical consta

def find_altitude(T):
    """This function calculates and prints the altitude above Earth's
    for a satellite with orbital period T (input in seconds)"""

    G = 6.67e-11 #Gravitational constant, units: m^3/kg/s^2
    M = 5.97e24  #mass of Earth, units: kg
    R = 6371e3   #radius of Earth, units: m

    h = (G*M*T**2/(4*PI**2))**(1/3)-R #calculate height of satellite i

    print('The correct altitude for an orbital period of {:d} seconds
```

In [2]:
```
t = 10000
find_altitude(t)
```

```
The correct altitude for an orbital period of 10000 seconds is 365775
0.1 meters, 3657.8 kilometers, or 2272.8 miles.
```

Output for 1b and 1c: Use code cells to carry out the desired calculations, and Markdown cells to present and discuss your results.

1b. Use your program to calculate the altitudes of satellites that orbit the Earth once a day (so-called "geosynchronous" orbit), once every 90 minutes, and once every 45 minutes. What do you conclude from the last of these calculations?

In [3]:
```
find_altitude(60*60*24)
find_altitude(90*60)
find_altitude(45*60)
```

```
The correct altitude for an orbital period of 86400 seconds is 358559
10.2 meters, 35855.9 kilometers, or 22279.9 miles.
The correct altitude for an orbital period of 5400 seconds is 279321.
6 meters, 279.3 kilometers, or 173.6 miles.
The correct altitude for an orbital period of 2700 seconds is -218155
9.9 meters, -2181.6 kilometers, or -1355.6 miles.
```

The altitude for a 45 minute period is negative, meaning that the orbital radius is less than the radius of the earth. Therefore, a 45 minute orbit is impossible.

1c. Technically a geosynchronous satellite is one that orbits the Earth once per *sidereal day*, which is 23.93 hours, not 24 hours. Why is this? And how much difference will it make to the altitude of the satellite?

Using the program from part a, and entering 23.93**60 = 1435.8 for T, we find that the altitude for a geosynchronous satellite is 35773762.3m = 35773.8km = 22233.5mi The 24 hour day is defined by the position of the sun relative to a point on Earth, which is determined not only by the rotation of the Earth about its axis, but also the revolution of the Earth about the sun. The sidereal day is defined only by the rotation of the Earth about its axis, this is the quantity of interest. The geosynchronous orbit corresponds to an altitude that is about 80km lower than that of the 24 hour orbit.

Rubric for Problem 1: 10 points

(a) 5 points: 2 points for completion, 3 points for the answers.

(b) 3 points: graded on the answers only.

(c) 2 points: graded on completion.

Problem 2: Perfect Hardboiled Eggs

[Adapted from Langtangen, Exercise 1.12. You may derive the formula in Physics 112 or Physics 89] As an egg cooks, the proteins first denature and then coagulate. When the temperature exceeds a critical point, reactions begin and proceed faster as the temperature increases. In the egg white the proteins start to coagulate for temperatures above 63 C, while in the yolk the proteins start to coagulate for temperatures above 70 C. For a soft boiled egg, the white needs to have been heated long enough to coagulate at a temperature above 63 C, but the yolk should not be heated above 70 C. For a hard boiled egg, the center of the yolk should be allowed to reach 70 C.

The following formula expresses the time t it takes (in seconds) for the center of the yolk to reach the temperature T_y (in Celsius degrees):

$$t = \frac{M^{2/3} c \rho^{1/3}}{K \pi^2 (4\pi/3)^{2/3}} \ln\left[0.76 \frac{T_0 - T_w}{T_y - T_w}\right].$$

Here, M, ρ, c, and K are properties of the egg:

- M is the mass,
- ρ is the density,
- c is the specific heat capacity, and
- K is the thermal conductivity.

Relevant values are

- $M = 64\ \mathrm{g}$ for a large egg (USA size XL: en.wikipedia.org/wiki/Chicken_egg_sizes),
- $\rho = 1.0378\ \mathrm{g\ cm^{-3}}$,
- $c = 3.7\ \mathrm{J\ g^{-1}\ K^{-1}}$, and
- $K = 5.4 \cdot 10^{-3}\ \mathrm{W\ cm^{-1}\ K^{-1}}$.

Furthermore,

- T_w is the temperature (in C degrees) of the boiling water, and
- T_0 is the original temperature (in C degrees) of the egg before being put in the water.

Suppose we want our eggs hard-boiled. Implement the formula in a program, set $T_w = 100\ \mathrm{C}$ and $T_y = 70\ \mathrm{C}$, and compute t for a large egg taken from the fridge ($T_0 = 4\ \mathrm{C}$) and from room temperature ($T_0 = 20\ \mathrm{C}$). Also compute the results for a small egg ($M = 42\ \mathrm{g}$).

Output for 2: When you run your code cell, it should produce the following text, with your numbers instead of the `TTT`, `MMM`, and `SSS` placeholders:

```
To hard-boil a large egg taken directly from the fridge, cook
it for TTT minutes (MMM min, SSS sec).
To hard-boil a small egg taken directly from the fridge, cook
it for TTT minutes (MMM min, SSS sec).
To hard-boil a large egg starting from room temperature, cook
it for TTT minutes (MMM min, SSS sec).
To hard-boil a small egg starting from room temperature, cook
it for TTT minutes (MMM min, SSS sec).
```

The `TTT` placeholders should be values in minutes to two decimal places. The `MMM` and `SSS` placeholders should be rounded to the nearest minute/second, with no decimal places. For example,

```
To hard-boil a large egg taken directly from the fridge, cook
it for 56.78 minutes (56 min 47 sec).
```

In [4]:
```python
import numpy as np

def find_time(T_initial, M_egg):
    """This function calculates and returns the time (in seconds)
    for the center of the yolk to reach the temperature T_yolk, given
    the initial egg temperature T_initial (in degrees Celsius). The ma
    of the egg (in grams) is also taken as an input."""

    T_water = 100 #temperature of boiling water, units: degrees Celsiu
    T_yolk  = 70  #temperature of cooked yolk, units: degrees Celsius
    rho = 1.0378  #density of egg, units: grams/cm^3
    c = 3.7       #specific heat capacity, units: Joules/grams/(degree
    K = 5.4e-3    #thermal conductivity, units: Watts/cm/(degrees Kelv

    prefactor = M_egg**(2/3)*c*rho**(1/3) / ( K*np.pi**2*(4*np.pi/3)**

    t = prefactor * np.log( 0.76 * (T_initial - T_water) / (T_yolk - T
    t_TTT = t / 60      # time needed in minutes
    t_MMM = int(t_TTT)  # time needed in minutes, rounded down to near
    t_SSS = t % 60      # time needed in addition to t_MMM in seconds

    return [t_TTT, t_MMM, t_SSS]

M_large = 64        #mass of a large egg, in g
M_small = 42        #mass of a small egg, in g
T_room = 20         #desired temperature for hard-boiled egg, in Celsi
T_fridge = 4        #temperature of fridge, in Celsius

print("To hard-boil a large egg taken directly from the fridge, cook i
print("To hard-boil a small egg taken directly from the fridge, cook i
print("To hard-boil a large egg starting from room temperature, cook i
print("To hard-boil a small egg starting from room temperature, cook i
```

```
To hard-boil a large egg taken directly from the fridge, cook it for
6.41 minutes (6min, 25sec)
To hard-boil a small egg taken directly from the fridge, cook it for
4.84 minutes (4min, 50sec)
To hard-boil a large egg starting from room temperature, cook it for
5.10 minutes (5min, 6sec)
To hard-boil a small egg starting from room temperature, cook it for
3.85 minutes (3min, 51sec)
```

Rubric for Problem 2: 10pts

Effort and completion: 4 points

Output/answer: 6 points (2 points for the formatting, 4 points for the values)

Problem 3: Estimating Half-Life

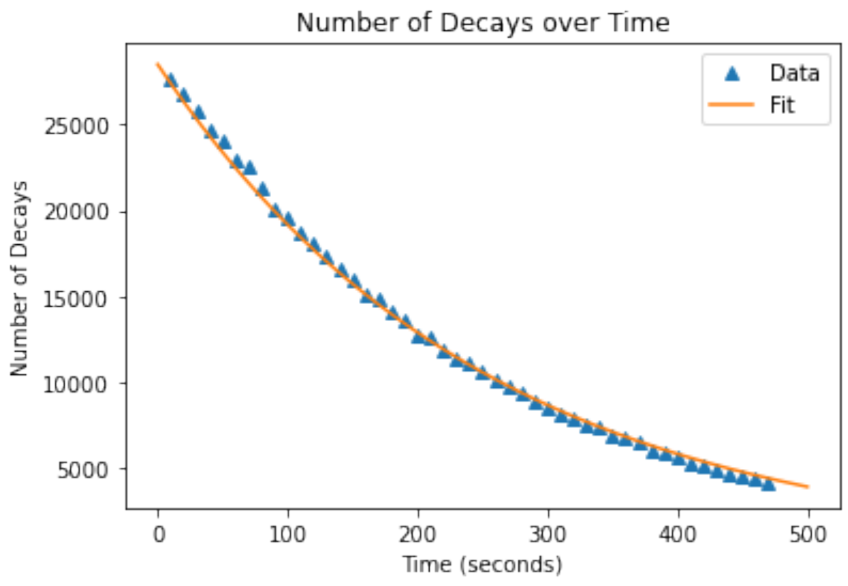
[Adapted from Ayars, Problem 0-3] The data in file Ba137.txt (https://raw.githubusercontent.com/celegante/code_chapter_0-_github/master/Ba137.txt) is actual data from a radioactive decay experiment (you should already have the file from the Workshop). The first column is the number of decays N, the second is the time t in seconds. We'd like to know the half-life $t_{1/2}$ of $^{137}\mathrm{Ba}$. It should follow the decay equation

$$N = N_0 e^{-\lambda t}$$

where $\lambda = \frac{\log 2}{t_{1/2}}$. Using the techniques you've learned from the lecture and workshop, load the data from the file Ba137.txt into appropriately-named variables. Experiment with different values of N and λ and plot the resulting equation on top of the data. (Python uses `exp()` to calculate the exponential function: i.e. `y = A*exp(-L*time)`) Don't worry about automating this process yet (unless you *really* want to!) just try adjusting things by hand until the equation matches the data pretty well. What is your best estimate for $t_{1/2}$?

Output for 3: When you run your code cell, it should produce a well-labeled plot with both the data and your curve of best fit. It should also print a message to the terminal which says, "My best estimate for the half life is x", where x is your estimate with units.

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt

counts, times = np.loadtxt("Ba137.txt", unpack = True) #reads data fro

#plots data
plt.plot(times, counts, '^')
plt.title('Number of Decays over Time')
plt.xlabel('Time (seconds)')
plt.ylabel('Number of Decays')

# Guesses for half life and initial amount
N0 = 28500
half_life = 175 #units: seconds

L = np.log(2) / half_life #lambda, units: inverse seconds

# Plots decay equation
t_values = np.linspace(0,500,500)
N_values = N0*np.exp(-L*t_values)
plt.plot(t_values, N_values)

plt.legend(['Data', 'Fit'])
plt.show()

# Prints half life
print('My best guess for the half life is {:d} seconds.'.format(half_l
```



```
My best guess for the half life is 175 seconds.
```

Rubric for Problem 3: 10pts

Effort/completion: 5 points

Output/answer: 5 points (1 point for the value of the estimated half life, 2 points for the correct plot excluding axis formatting, 2 points for axis formatting).