In [ ]:
```
# Initialize Otter
import otter
grader = otter.Notebook("lab02.ipynb")
```

1 E7: Introduction to Computer Programming for Scientists and Engineers

1.1 Lab Assignment 2

For each question, you will have to fill in one or more Python functions. We provide an autograder with a number of test cases that you can use to test your function. Note that the fact that your function works for all test cases thus provided does necessarily guarantee that it will work for all possible test cases relevant to the question. It is your responsibility to test your function thoroughly, to ensure that it will also work in situations not covered by the test cases provided

In [ ]:
```
# Please run this cell, and do not modify the contets
import math
import numpy as np
import matplotlib.pyplot as plt
#plt.style.use('fivethirtyeight')
#plt.rcParams["figure.figsize"] = (20,7)
%matplotlib inline

import hashlib
def get_hash(num):
    """Helper function for assessing correctness"""
    return hashlib.md5(str(num).encode()).hexdigest()
```

1.2 Question 1

Write a Python function that calculates the roots of a quadratic equation $ax^2 + bx + c = 0$ of unknown x, using the quadratic formula below.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The function output should be a two element tuple. The first element, `elem_1` ,should be the root computed by setting the $\pm$ to $+$, and the second element, `elem_2` ,should be the root computed by setting the $\pm$ to $-$. You may assume $b2 - 4ac \geq 0$.

```
In [ ]: def solve_quadratic(a, b, c):
            """
            Arguments
            ---------
            a : coefficient of x**2
            b : coefficient of x
            c : constant

            Returns
            -------
            elem_1 : larger root of quadratic equation
            elem_2 : smaller root of quadratic equation
            """
            discriminant = ...
            elem_1 = ...
            elem_2 = ...
            return elem_1, elem_2
```

```
In [ ]: solve_quadratic(6,-17,12)
```

```
In [ ]: grader.check("q1")
```

1.3 Question 2.0

Carbon dating is a method for determining the age of an object containing the radioisotope carbon-14. Since carbon-14 is subject to radioactive decay, the number of carbon-14 atoms in a sample will decay over time according to the following exponential decay law:

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

where N_0 is the number of atoms of the isotope in the original sample, $N(t)$ is the remaining number of atoms after a time t, λ is the expoential decay constant and t is the time.

Write a function that calculates the remaining fraction of carbon-14 in a sample after a certain amount of time (in years). The remaining fraction is defined as the ratio of the number of atoms at the present time ($N(t)$) to the initial number of atoms (N_0). You can re-arrange the decay formula above to find an expression for this ratio.

Assume $\lambda = 0.00012097\ \text{years}^{-1}$.

```
In [ ]: def c14_dating(time):
            ratio = ...
            return ratio
```

```
In [ ]: c14_dating(10000)
```

```
In [ ]: grader.check("q2.0")
```

1.4 Question 2.1

Now create a more general version of the function in Q2.0 and name the new function `exp_decay`. This new function will perform the same calculation as `c14_dating` but include a default argument for λ, the expoential decay constant. Call the argument for λ `decay_constant`. Set the default value for `decay_constant` equal to the exponential decay constant for c-14.

Write your function in the cell below (that contains the ellipsis `...`).

Your function should be in the form:

```
def exp_decay(...):
    ratio = ...
    return ratio
```

```
In [ ]: ...
```

```
In [ ]: exp_decay(10000)
```

```
In [ ]: grader.check("q2.1")
```

1.5 Question 3.0

Write a Python function that computes the Euclidean distances between two sets of points. The x-coordinate for the first point will be stored in a column vector `X_1`, the y-coordinate for the first point in `Y_1`, the x-coordinate for the second point will be stored in a column vector `X_2`, the y-coordinate for the second point in `Y_2`. Note that, for a single pair of points, (x_1, y_1) and (x_2, y_2), the Euclidean distance can be obtained from the distance formula:

$$\text{Euclidean distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

```
In [ ]: def euclidean_distance(x1, y1, x2, y2):
            """
```

```
    Arguments
    ---------
    x1 : The x-coordinates for the first set of points
    y1 : The y-coordinates for the first set of points
    x2 : The x-coordinates for the second set of points
    y2 : The y-coordinates for the second set of points

    Returns
    -------
    distance : Euclidean distance between points
    """
    x = ...
    y = ...
    distance = ...
    return distance
```

```
In [ ]: euclidean_distance(4,5,3,7)
```

```
In [ ]: grader.check("q3.0")
```

1.6 Question 3.1

Now that you have written a function to calculate the Euclidean distance between two points, recreate the function using default arguments of 0 for all inputs. Name your function `euclidean_distance_with_defaults`. The arguments should still be named `x1`, `y1`, `x2`, `y2`. Your function

Write your function in the cell below (that contains the ellipsis `...`).

Your function should be in the form:

```
def euclidean_distance_with_defaults(...):
    x = ...
    y = ...
    distance = ...
    return distance
```

```
In [ ]: ...
```

```
In [ ]: euclidean_distance_with_defaults(1,1)
```

```
In [ ]: grader.check("q3.1")
```

1.7 Question 4

A ball is launched in the air with initial velocity v_o from an initial position of coordinates (x_0, y_0) as illustrated below:

Figure 1: Ball launching at angle θ from the x-axis with inital velocity v_0

The ball is launched at an angle θ (expressed in degrees) from the x-axis. The equations describing the motion (i.e. the position of the ball $(x(t), y(t))$ as a function of time t) of the ball are:

$$x(t) = x_0 + v_0 t \cos(\theta)$$
$$y(t) = y_0 + v_0 t \sin(\theta) - \frac{1}{2} g t^2$$

where g is the gravitational acceleration ($g = 9.81 m/s^2$). Since θ is given in degrees, you will want to convert it first into radians before using the `sin` or `cos` functions.

Question 4.1: Write a function that calculates the time at which the ball reaches the ground.

Note that when the ball hits the ground, $y = 0$. Also, before writing your code, don't forget to use pen and paper to manipulate the equation(s) above and get a mathematical expression for time.

```
In [ ]: # run this cell to create a global variable g, but make sure you don't have any other global va
        g = 9.81
```

```
In [ ]: def proj_time(y0, v0, theta):
            b = ...
            time = ...
            return time
```

```
In [ ]: proj_time(0, 15, 40)
```

```
In [ ]: grader.check("q4.1")
```

Question 4.2: Write a function that calculates the horizontal distance traveled by the ball before it reaches the ground. You can use the function `proj_time` and the equations of motion.

```
In [ ]: def proj_distance(y0, v0, theta):
            time = ...
```

```
    dist = ...
    return dist
```

```
In [ ]: grader.check("q4.2")
```

```
In [ ]: import matplotlib.image as mpimg
        img = mpimg.imread('resources/animal.png')
        imgplot = plt.imshow(img)
        imgplot.axes.get_xaxis().set_visible(False)
        imgplot.axes.get_yaxis().set_visible(False)
        print("Congrats on finishing lab 2!")
        plt.show()
```

To double-check your work, the cell below will rerun all of the autograder tests.

```
In [ ]: grader.check_all()
```

1.8 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

Make sure you submit the .zip file to Gradescope.

```
In [ ]: # Save your notebook first, then run this cell to export your submission.
        grader.export()
```