Iteration II: Simple Dynamics with the 'while' Loop

In this exercise you will use the `while` statement to loop over a set of floating point values to simulate an object falling from an aircraft. You will learn about a simple Python data structure called a list. You will also learn about Python modules, and in particular how to use the `matplotlib` module to create a graph.

An object (such as a bag of flour, or an army field ration) is to be dropped from an airplane, with the hope of hitting a target on the ground. You are to write a Python script which will compute the position of the bag as it falls, using a `while` loop to step over short periods of time.

We will neglect air resistance for now (we will include it in a later exercise). If the altitude from which the object is released is A (measured in feet, since those are the units used by aircraft altimeters) then the altitude at t seconds after the release is given by the formula

$$y = A - \tfrac{1}{2}\,(32.17)\,t^2\,,$$

where 32.17 ft/sec^2 is the acceleration due to gravity near the surface of the Earth. The forward position of the object t seconds after the release is given by

$$x = x_0 + v_0\,t$$

where x_0 is the position when the object is released, and v_0 is the forward speed of the aircraft at the time of release, measured in feet per second. For simplicity we can assume $x_0 = 0$ in what follows.

The `while` loop:

The `for` loop repeats the same set of instructions a fixed number of times. In contrast, the `while` loop repeats the instructions until a given condition is met, such as the object hitting the ground. A common use is to take small steps forward in time, repeating the same calculations again and again, as might be done to plot a graph.

Here is an example of how to compute the points on the curve $y(t) = at^2 + bt + c$ from $t = 0.0$ seconds, every 0.1 seconds, up to a maximum of 30.0 seconds, or when y is no longer positive:

```
t = 0.0
dt = 0.1
while  t <= 30.0 and y > 0.0 :
      y = a*t*t + b*t + c
      print(t, y)
      t = t + dt
```

Note the colon at the end of the condition, just as in a `for` or `if` statement. Keep in mind that it is possible to make the loop repeat endlessly (what's called an "infinite" loop) if the condition for stopping is never met. In the code above it is very important to update the time in the t variable as the last part of each step, or the result would be an infinite loop.

Lists and Arrays

A table of numbers is not the best way to save or display this kind of data. A graph would be much easier to read and understand, so we will use the `matplotlib` module to create a graph. To do so, your script will have to save all the values calculated into "list" variables. A list is simply an ordered set of values, where you can select any element in the list by its numerical position in the list. Here is a simple example.

```
heights = [ 5.6, 5.4, 6.2, 6.5, 5.9, 6.3 ]
print( heights[4] )
```

The first line creates a list which has 6 values. The second line selects out the 5th item in the list (because Python starts counting from zero, not one). Try it.

The `range()` function we used previously for the `for` loop actually creates a list of values. Here's a simple example:

```
someNumbers = range(0,11,2)
print(someNumbers[3])
```

A list of numbers is somewhat like a variable on a chalk board which has a subscript, and the number in square brackets is the subscript number. Other computer languages use the same concept, but call it an "array".

Python Modules

In Python a module is a file containing definitions of variables and functions which can be loaded into a script and used immediately. This is a powerful feature which makes it easy to extend the capabilities of your own scripts. The simplest example is the '`math`' module, which contains basic mathematical functions for taking square roots or computing trigonometric functions. On its own, Python does not know how to do these things. But all you have to do is load the '`math`' module, and then use these functions in whichever calculations they are needed.

Here is a simple example. Suppose you have the x and y components of a vector saved in variables `V_x` and `V_y`, and you want to compute the magnitude and direction of that vector. The magnitude is the square root of the sum of the squares, and the direction is the inverse tangent of the ratio of the y component to the x component. In Python you could compute these with:

```
import math
V_mag = math.sqrt(V_x*V_x + V_y*V_y)
```

```
V_dir = math.atan(V_y/V_x) * 180 / math.pi
```

The last line uses the mathematical constant π to convert the angle from radians to degrees. There is actually an ambiguity of the resulting direction – it could be off by 180°. You an easily get around that by instead using the function `math.atan2(V_y,V_x)`, which automatically accounts for the proper quadrant.

You only have to import a module once, and you can do so anywhere in your script before you want to use something from that module, but it's generally a good idea to import modules at the beginning of a script to show anyone reading your code which modules the script uses.

Making a Graph with MatPlotLib

A useful Python module which makes it easy to plot graphs, both on the screen and saved to an image file, is called "`matplotlib.pyplot`" because it provides the same functionality as the MATLAB plot library. To use it you have to first install the module (only once) on your computer, using the PIP utility. As you learned in the last lesson, open up a command prompt window (for your operating system, not the IDLE shell window) and give the command:

```
>    pip3 install matplotlib
```

In your script you then make use of the module by "importing" it with the `import` statement. In your script add the line:

```
import matplotlib.pyplot as plt
```

The second part lets you refer later to this module simply as "plt", which will save you a lot of typing.

The next step is to have your script compute the values to be plotted and save them in two lists (arrays), one for the horizontal values, and the other for the corresponding vertical values. Before you get started, you create two empty arrays to hold the values. Then each time through the loop you use the `array.append()` function to add another value to each list. Here is an example, building on the earlier one, to plot the function $y(x) = ax^2 + bx + c$, for values of x between 0.0 and 30.0 in steps of 0.1:

```
a = 5;    b=-3;   c=1
x = 0.0
dx = 0.1
x_ray = []
y_ray = []
while  x <= 30.0 :
      y = a*x*x + b*x + c
      print(x, y)
      x_ray.append(x)
      y_ray.append(y)
      x = x + dx
```

After the loop completes, the arrays will be filled with the appropriate numerical values. It is then straightforward to make the graph like this:

```
plt.figure()                                    # starts a new figure
plt.suptitle("main title here")                 # Overall title
plt.xlabel('X axis label here (units)')
plt.ylabel('Y axis label here (units)')
plt.plot(x_ray,y_ray)                           # creates the plot
```

This creates the plot, but does not display it. You can either save the plot to an image file, or display it on the screen, or both, using one or both of the following:

```
plt.savefig("imageFileName")
plt.show()
```

The first line saves the graph to an image file named `imageFileName.png`. The second line opens a new window to display the graph, and pauses your script. The script will continue right after where it stopped once you close the graph window.

Assignment

Your goal for this exercise is to write a Python script which simulates a falling object released from an airplane in level flight at constant speed.

1. **Input:** Your script should first prompt for and read the altitude of the aircraft (in feet) when the object is dropped. Next, it should prompt for and read the airspeed of the aircraft when the object is dropped, in knots (nautical miles per hour) since those are the units used on airspeed indicators in most small aircraft. This will need to be converted into the initial horizontal speed `vx0`, in feet per second. To do this it will help to know that there are 6076 feet in a nautical mile.

   Your script should assume that the initial horizontal position at the time of the drop, `x0`, is zero. Your script should echo the values of all variables as you read them in. For some representative input values for a Cessna 150 try an altitude of 200 feet, and an airspeed of 60 knots.
2. **Calculations:** Your script should loop over a time interval of up to 30 seconds, with a step size of 0.1 seconds, and determine the x and y positions of the object at each time step. The loop should exit early if the altitude of the object becomes zero or less than zero (meaning that the object has struck the ground).
3. **Output:** At each time step (each time through the `while` loop) you should print the time, followed by the position of the object (horizontal first, then vertical), all on the same line. If your program gets all the way to 30.0 seconds and the object has not yet struck the ground, then it should print a warning message saying that the calculation did not finish. Do not print this warning if the object reaches the ground, but do not go over 30.0 seconds.
4. **Graph:** Once all computations have been completed, your script should display a graph of the trajectory of the dropped object, plotted as y-vs-x. Your graph must

have appropriate labels for each axis (with the correct units) and an overall title for the graph (the "`suptitle`" demonstrated above).