Introduction to NetLogo
NetLogo is a programming language for simulating interactions between "agents" in a suitable environment (such as the interactions between ants in a colony, or sheep and wolves in a field). This approach to modeling makes it possible to examine how the simple behavior of individuals results in the complex collective behavior of a group. The scope of the language as a modeling tool encompasses biology economics, social sciences, physics and chemistry.

NetLogo can be used in a number of ways. First, we can explore the large number of existing models of emergent phenomena that come with NetLogo, to see the types of phenomena that NetLogo is well suited to modeling. We can conduct virtual experiments to adjust the various model parameters and observe the response of the system. By so doing we can test whether the model fits with observations in the real world, and whether it predicts unexpected behavior. We can also modify existing models, for example, by adding a new species to a predator/prey model. The ultimate goal, however, is to learn the programming language in sufficient detail to create our own models which you might use as demos or labs in your classes. In this first lab we will start by familiarizing ourselves with the NetLogo interface and some of the basic commands for controlling the "agents" and their environment. If you need help understanding any of the commands there is a helpful User Manual available under the help menu. The programming guide and primitives dictionary will be useful references for you in these workshops. Lets start with the first two tutorials provided in the NetLogo User Manual.

Tutorial #1: Models
Complete Tutorial #1: Models, from the NetLogo User Manual. This tutorial explores one particular model form the models library and serves as an introduction to NetLogo's graphical user interface and the elements that allow you to control the model. The tutorial poses questions and asks you to make predictions.

Tutorial #2: Commands
Complete Tutorial #2: Commands, from the NetLogo User Manual. This tutorial is an introduction to the different ways to interactively control your modeling environment using commands. This is an important precursor to writing procedures.

Writing Procedures
You have seen how to control NetLogo Models using sliders and buttons and have learned how to use the Command Center and agent monitors to interact with patches and turtles. Now you will learn how to control the interactions of patches and turtles by writing procedures. Procedures are groups of NetLogo *commands* or *reporters. Commands* are actions that patches or agents carry out. *Reporters* are commands that return a value that might be acted on by another command. Together these are referred to as NetLogo *primitives.* An example of a simple procedure is given below:

```
to wiggle
        ask turtles [
                left random 40
                right random 40
                forward 1 ]
end
```

In this procedure ask, left, right and forward are commands, and random 40 is a reporter that returns an integer between 0 and 39. Note that a procedure starts with a to and end with an end. The next tutorial shows how to write procedures. Note this tutorial is different than tutorial number 3 in the NetLogo User manual (They changed the tutorial in the latest version of NetLogo.

Tutorial 3: Procedures
Complete the tutorial Procedures from the Origins Lab web page. Work through this tutorial carefully. As you learn a new command or reporter take the time to play around with it so that you fully understand how to use it and what it can do. Resist the urge to copy and paste code from the Tutorial to your program. As you type it yourself you will reinforce the logic of each command. At the end of the tutorial you should be well on your way to writing your own programs.

Extension to the Model
In the tutorial you created a program that models how an organism can find the maximum of some quantity by following the direction of the local gradient (ie the steepest slope). As the program is written the turtles are very successful at finding local maxima, but struggle to find the global maximum, unless the smoothness is set so high that there is only one maximum! At the end of the tutorial you are asked to think about ways to get turtles to find the global maximum. There are a couple strategies you can take here. One is simply to have the turtles jump to a random spot on the screen if they find themselves at a local maximum that is not near enough to the highest value. Modify your program to implement this strategy. A philosophical problem with this strategy from the point of view of the emergence of order based on local interactions is that the turtles may not have a reason to know they are not at the global maximum. The following guiding questions will allow you to investigate a more natural strategy, while learning some new NetLogo features. Save your work in a NetLogo file with naming convention LastName_Firsname_Lab1_nlogo.

Finding Maximum Fitness
You may have heard about the evolutionary concept of a fitness landscape. The idea is that phenotypic variables such as pigment, or size form variables in a morphological landscape, whose elevation is "fitness". The fitter the organism the more likely it is to produce offspring. Offspring of organisms may mutate – appearing in a different part of the landscape from their parent. Those that emerge at a higher elevation are fitter and will breed more successfully. Over time organisms evolve until most are at a peak of the landscape. This method of using evolution, rather than gradients, to find local maxima can be extremely efficient. It is also easy to implement an algorithm that can find global maxima.

1. Make the following changes to your interface:
   (a)   Change the graphics window so that the max-pxcor and max-pycor are both 50 and the patch size is 4.
   (b)   Change the smoothness slider so it has a maximum of 50.
2. One problem with the way the previous model smoothes the landscape is that smoothing not only makes the landscape less rough, but it also diminishes the difference in elevation between the highest peak and deepest valley. The following changes to the setup-patches procedure solve this problem:
   (a)   After the command that finds the highest and lowest elevation, rescale the

elevation variable so that lowest is 0 and highest is 100. (Note: You cannot simply redefine highest and lowest – you have to rescale the elevation variable for all the patches by using the ask patches command and a suitably chosen linear rescaling.)

    (b)   After this command recolor the patches so that the color scale ranges between `0` and `100` rather than 1000 to 9000.

3. Make the following changes to the `go` procedure:
    (a)   Remove out references to movement – the turtles do not move in this program.
    (b)   Call two new procedures: `birth` and `death`. You will write these procedures.

4. Write the `birth` procedure so that it has the following properties:
    (a)   At each time step a turtle hatches one offspring with a probability proportional to its elevation (or fitness). (Any turtles that happen to be at zero elevation should not reproduce and any turtles who are at elevation 100 should always reproduce.)
    (b)   Hatch new offspring so that they have the same color as their parent but scaled according to their fitness. Let high fitness be a darker shade than low fitness (so that turtles can be seen on the landscape).
    (c)   Hatch new offspring at a distance `mutation-rate` from their parent and have them face in a random direction. Define `mutation-rate` with a slider ranging from 0 to 10 on the interface.

5. Write the `death` procedure in a way that keeps the population size *roughly* constant. (for example, kill off turtles randomly if the population is greater than the starting number).

6. Add a new plot and a new monitor to your interface, showing the average fitness of the turtles as it changes over time.

7. Now use your new model to investigate how the turtles find maxima. You should see that turtles find their way to the global maximum fairly often, but not always. Experiment with different landscapes. Does the smoothness of the landscape affect the average fitness? Experiment with the `mutation-rate` slider. How does increasing `mutation-rate` affect average fitness? When turtles get trapped on a local maximum, you can often nudge them over to the global maximum by increasing `mutation-rate` for a while and then decreasing it again. Can you think of evolutionary pressures that might increase mutation rates like this from time to time?

When you are finished your model copy and paste it into the Origins file space on MASU.