Lab: Programming Key Concepts

```{r lab2_setup_guard, include=FALSE}
if (!exists("a")) a <- 2
if (!exists("b")) b <- 3
if (!exists("c")) c <- "hello"
if (!exists("r")) r <- 0.6
if (!exists("K")) K <- 100
if (!exists("n0")) n0 <- 10
knitr::opts_chunk$set(error = TRUE, warning = TRUE, message = TRUE)
```
```{r lab_setup, include=FALSE}
knitr::opts_chunk$set(comment = NA)
```

Lesson Overview

The goals of this modules is to refine your knowledge of the basic, core concepts of programming that transcend languages, how they fit together, and how you can use them to become a better scientist. We will practice these concepts by modeling the logistic growth of a population.

Conservation/ecology Topics

- Simulate population dynamics under logistic growth, and how a catastrophes alter those dynamics.

Computational Topics

By the end of this R lesson, you will be able to:

- Gain familiarity with the 6 of the 7 core elements shared by all programming languages.
- Use R to write simple functions that use these core elements.
- Make and save simple plots using basic plots.

We encourage you to make liberal use of your fellow students, GE, and professor as we proceed through these modules. We will work in pairs.

Review: The Seven Core Concepts

As noted by Greg Wilson (the founder of Software Carpentry), every programming language shares seven core elements:

1. Individual things (the number 2, the character 'hello')
2. Commands that operate on things (the + symbol, the `length` function)
3. Groups of things (R vectors, matrices, lists, dataframes, and arrays)
4. Ways to repeat yourself (for and while loops, apply functions)
5. Ways to make choices (if and try statements)
6. Ways to create chunks (functions, objects/classes, and packages)
7. Ways to combine chunks (piping) (for next week)

TIP reminder: Some helpful R studio shortcuts
1. Run the current line of selection
   - Windows: `Ctrl-Enter`
   - Mac: `Command-Enter`
2. Source the entire script
  - Windows: `Ctrl-Shift-Enter`
  - Mac: `Command-Shift-Enter`

Question 1: Population dynamics

Logistic growth of a population: Throughout this lesson, we will successively build towards calculating the logistic growth of a population of bees in a meadow (or some less exciting vertebrate, if you prefer).

A commonly used discrete time equation for logistic population growth is

$$n(t + 1) = n(t) + r*n(t) [1 - n(t) / K]$$

where n(t) is the population size at time t, r is the net per capita growth rate, and K is the carrying capacity of the habitat.

To get started, write R expressions that do the following:

- 1a. Create variables for `r`, `K`, and `n0`, setting these equal to 0.6, 100, and 10, respectively.

```{r}
# your code here
r <- 0.6
K <- 100
n0 <- 10
```

Question 2: Using operators and functions

- 2a. Create the variable `n1` (n at t=1) and calculate it's value using the logistic growth equation Do the same for `n2` and `n3`.

n(t + 1) = n(t) + r*n(t) [1 - n(t) / K]

```{r}
# your code here
n1 <- n0 + r*(n0)*(1 - n0 / K)
n2 <- n1 + r*(n1)*(1 - n1 / K)
n3 <- n2 + r*(n2)*(1 - n2 / K)
```

- 2b. Check the type of `n2` - what is it?

```{r}
# your code here

class(n2)
```

- 2c. Modify your calculations for `n1`, `n2` and `n3` so that these values are rounded to the nearest integer (so no decimal places). If you have forgotten the name of the function that rounds numbers, try googling it (or look back at the lecture demo above). If you are unsure of the arguments to the function, check the help file using ?

```{r}
# your code here

round(n1, 0)
round(n2, 0)
round(n3, 0)
```

Question 3: Storing and indexing data

- 3a. Create a vector where `n0`, `n1`, `n2` and `n3` are stored, instead of as separate individual variables by creating an empty vector using the syntax `n <- vector("numeric", 4)`, and then assigning each index. You could get the same result using `c()`, but practice using `vector()` and indexing instead.

```{r}
# your code here
n <- vector("numeric", 4)
n[1] <- n0
n[2] <- round((n[1] + r*(n[1])*(1 - n[1] / K)),0)
n[3] <- round((n[2] + r*(n[2])*(1 - n[2] / K)),0)
n[4] <- round((n[3] + r*(n[3])*(1 - n[3] / K)),0)
```

```

- 3b. Get the first and last values in the vector, calculate their ratio, and print out "Grew by a factor of" followed by the result.

```{r}
# your code here
ratio <- n[1] / n[4]
print(paste("grew by a factor of", ratio))

```

- 3c. Extract the last value of your n vector in two different ways: first, by using the index for the last item in the vector, and second, with out "hard coding" the index and instead using a function to work out the index for the last element. HINT: the `length()` function may be useful.

```{r}
# your code here
n[4]
finaln <- function(n){
  last_in <- n[length(n)]
  return(last_in)
}
```

Question 4: Storing data and logical indexing

- 4a. Pre-allocate an vector called n containing 100 blank space (i.e. 0s) as if we were going to fill in 100 time steps.

```{r}
# your code here
n <- vector("numeric", 100)
```

- 4b. Imagine that each discrete time step actually represents 1 day. Create an vector `t` storing 100 time step from 2 to 100. For example, `t[1]` should be 2 `t[2]` should be 3, etc.

```{r}
# your code here
t <- c(2:100)
t[2]
```

- 4c. Use the logistic growth equation to fill in the first 5 elements of n by hand. Assume the initial population size at t=1 is 10.

Logistic growth equation:
$n(t + 1) = n(t) + r*n(t)[1 - n(t) / K]$

```{r}
# your code here
n0 <- 10
n[1] <- n0
n[2] <- round((n[1] + r*(n[1])*(1 - n[1] / K)),0)
n[3] <- round((n[2] + r*(n[2])*(1 - n[2] / K)),0)
n[4] <- round((n[3] + r*(n[3])*(1 - n[3] / K)),0)
n[5] <- round((n[4] + r*(n[4])*(1 - n[4] / K)),0)

n[1]
n[2]
n[3]
n[4]
n[5]
```
```

Rather painful! Hard to imagine we could fill in the rest without making a mistake with our indexing. In the next Question of the lab we will learn how to repeat ourselves without copy-pasting like in the above.

- 4d. Use logical indexing to extract the value of `n` corresponding to a `t` of 3.

```{r}
# your code here
n[t==3]

```

Question 5: Using loops to repeat calculations

Exciting next step, let's get smart about our calculations of `nt`. Building on what you did in Question 5, do the following:

- 5a. Write a for loop to fill in the values of `nt` for 100 time steps. Loop over the values of the t, and use each step vector to index the vector `n`. (Why does the t vector start at 2 and not at 1?)

```{r}
# your code here
for(k in 2:100){
  n[k] <- round((n[k-1]) + r*((n[k-1]))*(1 - (n[k-1]) / K),0)
  print(n[k])
}
```

- 5b. Plot the vector `n`. HINT: You will want t to start at 1 for the plot so it can include n(1) which we set above but did not include in the for loop because you need to start with an initial population before growing logistically.

```{r}
# your code here
t_new <- c(1,t)
plot(x=t_new, y=n, xlab="time", ylab="population size")

```

- 5c. Play around with the values of `r` and `K` and see how it changes the plot. What happens if you set `r` to 1.9?

```{r}
# your code here
for(k in 2:100){
  n[k] <- round((n[k-1]) + r*((n[k-1]))*(1 - (n[k-1]) / K),0)
  print(n[k])
}
r <- 1.9
K <- 100
plot(x=t_new, y=n, xlab="time", ylab="population size")

```

- 5d. What happens if you set `r` to 3?

```{r}
# your code here
r <- 3
plot(x=t_new, y=n, xlab="time", ylab="population size")
```

Question 6: Making the model stochastic with an if statement

Let's introduce some element of randomness into our logistic growth model to better represent nature. We'll model a simple "catastrophe" process, in which a catastrophe happens in 10% of the time steps that reduces the population back down to the size at

n0. For example, the bees in our meadow have some probability of being sprayed by herbicide that drifts from nearby timber plantations which would kill individuals directly and through starvation (no flowers left). Build on your code from Question 4 into the box below, and do the following:

- 6a. Create a variable called `cata`, for catastrophe, that will be `TRUE` if a catastrophe has occurred, and `FALSE` if it hasn't. A simple way to do this is to generate a random number using `runif(1)` (draw from a uniform 0,1 distribution once, see ?runif), which will give you a random number between 0 and 1. Check whether this number is less than 0.1 - this check will be `TRUE` 10% of the time.

```{r}
# your code here
cata <- runif(1) < 0.1
cata
```

- 6b. Using your logical variable `cata`, add an if statement to your for loop that checks whether `cata` is true in each time step. If it is true, set the population back to the size at n[0]. Otherwise, perform the usual logistic growth calculation. HINT: `cata` will need to be within the for loop so it change values each iteration.

```{r}
# your code here
for(k in 2:100){
  cata <- runif(1) < 0.1
  if (cata == TRUE){
    n[k] <- n0
  }else{
  n[k] <- round((n[k-1]) + r*((n[k-1]))*(1 - (n[k-1]) / K),0)
  print(n[k])
  }}
```

- 6c. Plot your results. Run the code again to see a different growth trajectory.

```{r}
# your code here
plot(x=t_new, y=n, xlab = "time", ylab = "population")
```

- 6d. Now that you have the vector `n`, count the number of time steps in which the population was above 50. Although you can do this with a for loop (loop through each value of `nt`, check if it is > 50, and if so increment a counter), you can do this in one line with a simple logical operation. HINT: If you take the sum of a logical vector (using `sum()`), it will give you the number of `TRUE` values (since a `TRUE` is considered to be a 1, and False is a 0).

```{r}
# your code here
sum(n > 50)
```

Question 7: Creating a logistic growth function

- 7a. Finally, let's turn our logistic growth model into a function that we can use over and over again. Let's start with writing a function to calculate n(t+1). It should take n(t), r, and K as agrguments and return n(t+1).

Reminder of the logistic growth equation:
 n(t + 1) = n(t) + r*n(t) [1 - n(t) / K]

```{r}
# your code here
logGrowth <- function(n_t,r,K){
  n(t+1) <- (n_t + r*(n_t))*(1 - n_t / K)
  return(n(t+1))
```

```
}
```

- 7b. Create function called `logistic_growth` that takes four arguments: `r`, `K`, `n0`, `p` (the probability of a catastrophe), and nsteps (the length of the t vector). Make `p` a default argument with a default value of 0.1. Have your function recreate your answer for question 7b above (simulate a catastrophe, grow the population if no catastrope occurs). Have your function return the `n` and `t` matrix.

- 7c. Write a nice comment describing what your function does. In addition, use comments with full sentences to describe the intention of each section of code.

```{r}
# your code here
logistic_growth <- function(r, K, n0, p=0.1, nsteps=100){
#creating an empty vector
  t <- 1:nsteps
  n <- vector("numeric",100)
#adding the probability of catastrophe into the function
for(k in 2:nsteps){
  cata <- runif(1) < p
  if (cata == TRUE){
    n[k] <- n0
  }else{
  n[k] <- round((n[k-1]) + r*((n[k-1]))*(1 - (n[k-1]) / K),0)
  }}
#returning the matrix of t and n by binding the vectors
  return(cbind(t,n))
}
cbind(t,n)

#The logistic_growth function calculates the exponential population growth of a population of bees ( or any other species of wildlife), including the chance that a catastrophe occurs and kills all of the population
```

- 7d. Call your function with different values of the parameters to make sure it works.

Store the returned results and make a plot from it.

```{r}
# your code here
plot(logistic_growth(0.5, 100, 10, p=0.7, nsteps=100), xlab = "Time", ylab = "Population Size")
```

- 7e. Explore different values of p and see how it changes the trajectory. Describe what happens.
ANSWER: ....
As the value of p increases, the number of times that the population size returns to n0 (or 0) increases, reflecting the increased probability that catastrophe happens and thus the population is wiped out.

Question 8 (Extra credit or graduate students)

** Analyze and simulate a discrete-time predator–prey system.**
1) simulate dynamics for two parameter sets,
2) visualize trajectories in time

We use logistic prey growth, a linear functional response, and linear predator mortality:
\[
\begin{aligned}
N_{t+1} &= N_t + r\,N_t\left(1-\frac{N_t}{K}\right) - a\,N_t P_t,\\
P_{t+1} &= P_t + e\,a\,N_t P_t - m\,P_t,
\end{aligned}
\]
where \(N_t\) = prey abundance, \(P_t\) = predator abundance, and parameters \

(r,K,a,e,m>0\).

- 8a: Use the parameter set below and initial conditions \(N_0=100,\;P_0=20\). Simulate for \(T=200\) steps. Make a time-series plot of \(N_t\) and \(P_t\)

```{r}

# define initial conditions

N_0 <- 100  #prey abundance at t = 0
P_0 <- 20  #predator abundance at t = 0

# set up the function
logistic_prey_growth <- function(r, K, a, e, m=0.1, nsteps=200){
# create an empty vector
time <- 1:nsteps
preyGrowth <- vector("numeric", nsteps)
predatorGrowth <- vector("numeric", nsteps)
#set initial conditions for new vectors
preyGrowth[1] <- N_0
predatorGrowth[1]<-P_0
# create a for loop to return function
for(t in 2:nsteps){
preyGrowth[t] <- preyGrowth[t] <- preyGrowth[t] + (r*preyGrowth[t]) * (1-
preyGrowth[t]/K) - a*preyGrowth[t]*P_[t]
predatorGrowth[t] <- predatorGrowth[t] + e*a*predatorGrowth[t]*predatorGrowth[t] -
m*predatorGrowth[t]
}
return(cbind(time,preyGrowth, predatorGrowth))
}
ppresults <- logistic_prey_growth(r = 0.5, K = 1000, a = 0.01, e = 0.1, m = 0.1, nsteps
= 200)
# show the data in a time series plot
plot(ppresults$time,ppresults$predatorGrowth, ppresults$preyGrowth) + geom_line()

```

- 8b: Repeat the simulation with **higher predator mortality** \(m=0.28\) (others unchanged). Compare the dynamics to part (a) including the visualization.

- 8c: What phenomenon of predator–prey dynamics does this illustrate?