

```

---
output:
  pdf_document: default
  html_document: default
---
## 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}
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`.

```
```{r}
n1 <- n0 + r * n0 * (1 - n0 / K) #here I am creating the equation in a format suitable for
r. It's basically saying at the population size at the first time stamp is equal to the
population at the initial size, plus growth rate time the initial size times 1 minus the
initial size over carrying capacity. proper formatting for equation is using parentheses, as
square brackets index, uses most recent generation in each sequence for time
n2 <- n1 + r * n1 * (1 - n1 / K)
n3 <- n2 + r * n2 * (1 - n2 / K)
```
```

```
```{r}
n1
n2
n3
```
```

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

```
```{r}
typeof(n2) #In R, when you see that something is of type "double", it means it's a numeric
value stored as a double-precision floating-point number - i.e., a standard real number
that can have decimals.
```
```

- 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}
round(n1)
round(n2)
round(n3)
```
```

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}
n <- vector("numeric", 4) #creating a numeric vector with 4 places
n[1] <- n0
n[2] <- n1
n[3] <- n2
n[4] <- n3
n
assigning each of those place holder using square brackets to call position
```

#square brackets assign position, remember just calling numeric and 4 will not assign
values, these variables need to be sorted
```

– 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}
value1 <- n[1]
value4 <- n[4]
ratio <- value1 / value4
ratio1 <- n[1]/n[4]
ratio1
print(paste("Grew by a factor of", ratio))
#it's making a sentence
```
```

– 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}
n[4] #first way, hard coding, straglit forward the square brackets!!!
n[length(n)] #second way, using length function to find last index, could this work to
call other positions?
```
```

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}
n <- vector("numeric", 100)
#since no positions were assinged (like n[1] above) all remain zero, woohoo and will be
filled later!
n[1:10]
```
```

- 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}
t <- seq(from = 2, to = 100, by = 1)
t[1:10]
length(t)
```

```

- 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}
r <- 0.6
K <- 100
n1 <- 10
n1 <- n0 + r * n0 * (1 - n0 / K)
n2 <- n1 + r * n1 * (1 - n1 / K)
n3 <- n2 + r * n2 * (1 - n2 / K)
n4 <- n3 + r * n3 * (1 - n3 / K)
n5 <- n4 + r * n4 * (1 - n4 / K)
```

```

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}
n[t == 3]
#what is n when t equals three
```
#straightforward, but remember this concept
```

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
r <- 0.6
n <- vector("numeric", 100)
n[1] <- 10
t <- 2:100
for (i in t) {
 n[i] <- n[i - 1] + r * n[i - 1] * (1 - n[i - 1] / K)
 #calculate each population based on previous one, so subtracting 1
```

```

}
n[1:10]
#n[i] <- n[i] + r * n[i - 1] * (1 - n[i - 1] / 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 <- 1:100
library(ggplot2)
ggplot(data = data.frame(t, n), aes(x = t, y = n)) +
  geom_line(color = "red") +
  labs(x = "Time step", y = "Population", title = "Logistic Growth over 100 Steps") +
  theme_minimal()
length(t)
length(n)
#why are lengths not matching up?

ggplot(data = data.frame(t, n), aes(x = t, y = n)) +
  geom_line()
#I had go update t to make t and n same length, check to see if this is what everyone did?
```

```

- 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}
r <- 1.9
n <- vector("numeric", 100)
n[1] <- 10
t <- 2:100
for (i in t) {
  n[i] <- n[i - 1] + r * n[i - 1] * (1 - n[i - 1] / K)
  #calculate each population based on previous one, so subtracting 1
}
t <- 1:100
library(ggplot2)
ggplot(data = data.frame(t, n), aes(x = t, y = n)) +
  geom_line(color = "red") +
  labs(x = "Time step", y = "Population", title = "Logistic Growth over 100 Steps") +
  theme_minimal()
#or
ggplot(data.frame(t, n), aes(x = t, y = n)) +
  geom_line()
...

```

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

```

```{r}
r <- 3
n <- vector("numeric", 100)
n[1] <- 10
t <- 2:100
for (i in t) {
 n[i] <- n[i - 1] + r * n[i - 1] * (1 - n[i - 1] / K)
 #calculate each population based on previous one, so subtracting 1
}
t <- 1:100
library(ggplot2)

```

```
ggplot(data = data.frame(t, n), aes(x = t, y = n)) +
 geom_line(color = "red") +
 labs(x = "Time step", y = "Population", title = "Logistic Growth over 100 Steps") +
 theme_minimal()
````
```

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 n_0 . 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}
cata <- runif(1) < 0.1
cata
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}
For loop with catastrophe check
r <- 0.6
t <- 2:100
for (i in t) {
 # Generate random catastrophe event (10% chance)
 cata <- runif(1) < 0.1

 if (cata) {
 # Catastrophe: reset population to initial size
 n[i] <- n0
 } else {
 # Normal logistic growth
 n[i] <- n[i - 1] + r * n[i - 1] * (1 - n[i - 1] / K)
 }
}
n[1:100]
````
```

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

```
````{r}
t <- 1:100
library(ggplot2)
ggplot(data = data.frame(t, n), aes(x = t, y = n)) +
 geom_line(color = "red") +
 labs(x = "Time step", y = "Population", title = "Logistic Growth over 100 Steps") +
 theme_minimal()
````
```

- 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}
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 arguments and return $n(t+1)$.

Reminder of the logistic growth equation:

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

```
```{r}
growth <- function(n, r, K) { #creating a function where the values of n, r, K can be
 changed
 growth_n <- n + r * n * (1 - n / K) # naming the equation, where n is the current
 population size, rate of growth, and K can all be altered. This is not exactly right, need
 to figure out how to grow it by value of one
 return(growth_n) #telling the growth function to return the equation growth_n when
 values are altered
}
growth(10, 0.6, 100) #example call to function
growth(11, 0.6, 100)
```
```

- 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 catastrophe 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}
nsteps <- length(t) #creating nsteps
nsteps
n <- vector("numeric", 100)
logistic_growth <- function(r, K, n0, nsteps, p = 0.1) {
 t <- 1:nsteps #t and n must be defined inside in functions environment, also nsteps is
 just like t but easier in place of creating a function
 n <- numeric(nsteps) #pre-allocate n vector
 n[1] <- n0 #set initial population size
 for(i in 2:nsteps){ #has to start at two, just like before in setting t in the first
 place
 cata <- runif(1) < p #generate random catastrophe event based on probability p which
 has now been defined
 if (cata) {
 # Catastrophe: reset population to initial size
 n[i] <- n0
 } else {
 # Normal logistic growth
 n[i] <- n[i - 1] + r * n[i - 1] * (1 - n[i - 1] / K)
 }
 }
}
return(data.frame(t = t, n = n)) #returning a data frame with both t and n, so they can
be plotted together. t = t is saying (left) the col of t is the same as values of t
calculated, and n = n is saying the col of n is the same as values of n calculated
```
```

```

}
logistic_growth(r = 0.2, K = 100, n0 = 10, nsteps = 20)
...

```

– 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}
set.seed(10)
#negative growth
res1 <- logistic_growth(r = -0.1, K = 100, n0 = 10, nsteps = 100, p = 0.1)
#moderate
res2 <- logistic_growth(r = 0.3, K = 100, n0 = 10, nsteps = 100, p = 0.1)
#faster growth
res3 <- logistic_growth(r = 0.2, K = 100, n0 = 10, nsteps = 100, p = 0.1)
Add a label for each simulation
res1$type <- "Negative growth" #creating labels for types of growth to plot
res2$type <- "Moderate growth"
res3$type <- "Fast growth"

Combine them all
all_res <- rbind(res1, res2, res3) #combining all results into one data frame

Plot
GrowthwCatastrophes <- ggplot(all_res, aes(x = t, y = n, color = type)) +
 geom_line() +
 labs(
 title = "Logistic Growth with Catastrophes",
 x = "Time (t)",
 y = "Population (n)",
 color = "Scenario"
) +
 theme_minimal(base_size = 14)
ggsave("Growth_Cata.pdf", plot = GrowthwCatastrophes, width = 6, height = 4)
...

```

– 7e. Explore different values of p and see how it changes the trajectory. Describe what happens.  
ANSWER: ....

```

```{r}
set.seed(10)

cata0 <- logistic_growth(r = 0.2, K = 100, n0 = 10, nsteps = 100, p = 0)

cata10 <- logistic_growth(r = 0.2, K = 100, n0 = 10, nsteps = 100, p = 0.1)

cata50 <- logistic_growth(r = 0.2, K = 100, n0 = 10, nsteps = 100, p = 0.5)
# Add a label for each simulation
cata0$type <- "No cata" #creating labels for types of growth to plot
cata10$type <- "10 Cata"
cata50$type <- "50 Cata"

# Combine them all
all_cata <- rbind(cata0, cata10, cata50) #combining all results into one data frame

```

```
# Plot
ggplot(all_cata, aes(x = t, y = n, color = type)) +
  geom_line() +
  labs(
    title = "Logistic Growth with Catastrophes",
    x = "Time (t)",
    y = "Population (n)",
    color = "Scenario"
  ) +
  theme_minimal(base_size = 14)
```

#with a low p value, no catastrophes will happen. As p increase chance of catastrophes goes up. a p of 0.5 means there is a 50% chance of catastrophe

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:

```
\[
\begin{aligned}
N_{t+1} &= N_t + rN_t\left(1 - \frac{N_t}{K}\right) - aN_t P_t, \\
P_{t+1} &= P_t + eaN_t P_t - mP_t,
\end{aligned}
\]
```

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}
N <- vector("numeric", 200)
P <- vector("numeric", 200)
N[1] <- 100
P[1] <- 20
T <- 2:200
r <- 0.5 # prey intrinsic growth rate
K <- 500 # prey carrying capacity
a <- 0.01 # predation rate, not sure what to set
e <- 0.1 # conversion efficiency, not sure what to set
m <- 0.1 # predator mortality rate

for (i in T) { #for the variable in T (set above just like before)
  N[i] <- N[i - 1] + r * N[i - 1] * (1 - N[i - 1] / K) - a * N[i - 1] * P[i - 1] #run
  through equation N, with new part added in
  P[i] <- P[i - 1] + e * a * N[i - 1] * P[i - 1] - m * P[i - 1] # and solving for P now
}
```

```
# Time vector
time <- 1:200 #making a time vector, but could also just modify T like before
```

```
# Plot
plot(time, N, type = "l", col = "blue", ylim = c(0, max(N, P)), #plotting for 200 values,
ylim ensure N and P stay the same length
      ylab = "Population", xlab = "Time", lwd = 2) #setting labels and lines
lines(time, P, col = "red", lwd = 2) #Adds a new line to the existing plot (does not
create a new plot).
```

```
...
```

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

```
```{r}
N <- vector("numeric", 200)
P <- vector("numeric", 200)
N[1] <- 100
P[1] <- 20
T <- 2:200
r <- 0.5 # prey intrinsic growth rate
K <- 500 # prey carrying capacity
a <- 0.01 # predation rate
e <- 0.1 # conversion efficiency
m <- 0.28 # predator mortality rate

for (i in T) {
 m <- 0.28
 N[i] <- N[i - 1] + r * N[i - 1] * (1 - N[i - 1] / K) - a * N[i - 1] * P[i - 1]
 P[i] <- P[i - 1] + e * a * N[i - 1] * P[i - 1] - m * P[i - 1]
}

Time vector
time <- 1:200

Plot
plot(time, N, type = "l", col = "blue", ylim = c(0, max(N, P)),
 ylab = "Population", xlab = "Time", lwd = 2)
lines(time, P, col = "red", lwd = 2)
```
```

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

```
```{r}
#Altering m causes the predator population to die off more quickly (red line) allowing the
prey population to grow more rapidly. Eventually, the prey population faces resource
limitations and tapers off to a stable population
```
```