Lab: Week 2

36-350 – Statistical Computing

Week 2 – Fall 2020

Name: Kimberly Zhang

Andrew ID: kyz

You must submit **your own** lab as a PDF file on Gradescope.

Control-Flow Constructs

Question 1

(5 points)

Notes 2A (3-4)

Write an `if-else` construct that prints the statement "a < A" if `"a" < "A"` is `TRUE` and "a >= A" otherwise.

```
if ("a" < "A") {
  print("a < A")
} else {
  print("a >= A")
}
```

```
## [1] "a < A"
```

Question 2

(5 points)

Notes 2A (6)

Repeat Q1, but using the `ifelse()` function.

```
print(ifelse("a" < "A", "a < A", "a >= A"))
```

```
## [1] "a < A"
```

Question 3

(5 points)

Notes 2A (3-4)

Write an `if` construct that, if there are matching elements of `u` and `v` (i.e., if `u[i] = v[i]` for any index i), prints the number of matching elements, and otherwise prints the string "There are no matching elements."

```r
set.seed(999)
u = sample(100,100,replace=TRUE)
v = sample(100,100,replace=TRUE)
matching = sum(u & v)
if (matching != 0){
  print(matching)
} else{
  print("There are no matching elements.")
}
```

```
## [1] 100
```

Question 4

(5 points)

Notes 2A (3-4)

Confirm the short-circuiting behavior of an `if-else` construct by (a) setting the variable `t` to the value 4, and (b) writing an `if-else` construct that first checks if `t` is greater than or equal to 4 and if so prints "t >= 4", then checks to see if `t` is greater than zero and if so prints "t > 0", and otherwise prints "t <= 0".

```r
t = 4
if (t >= 4)
{
  print("t >= 4")
} else if (t > 0) {
  print("t > 0")
} else {
  print("t <= 0")
}
```

```
## [1] "t >= 4"
```

Looping

Question 5

(5 points)

Notes 2B (2-3)

Write a `for()` loop to add, elementwise, the vectors `x` and `y` defined below. Place each sum into the vector `x.plus.y` (which you can initialize using, e.g., `x.plus.y = rep(NA,4)`, where `NA` means "not available" or missing), then after the `for()` loop, display `x.plus.y`.

```r
x = c(1,2,3,4)
y = c(-2,2,-3,3)
x.plus.y= rep(NA, 4)
for (ii in seq_along(x)){
  x.plus.y[ii]= x[ii] + y[ii]
}
x.plus.y
```

```
## [1] -1  4  0  7
```

Question 6

(5 points)

Notes 2A (3-4) and 2B (2-3)

Write a `for()` loop to sum the natural logarithms of all elements of the vector `z`, defined below, that are positive. (Use `if` to check for positivity.) Display the result. Show that you can perform the same task without a `for()` loop, by utilizing logical-based vector subsetting and one call to `sum()`.

```
z = c(-5,1,2,-4,3,4,-3,6)
res= 0
for( ii in seq_along(z)) {
  if (z[ii] > 0) {
    res= res + log(z[ii])
  }
}
res
```

```
## [1] 4.969813
```

```
sum(log(z[z>0]))
```

```
## [1] 4.969813
```

Question 7

(5 points)

Notes 2B (5)

Write a `while()` loop that computes the sum of the first 100 positive integers. (Set the variable `s` equal to zero, then increment its value with each loop.) Then display the result. (It should be 5050.)

```
s= 0
res = 0
while (s <= 100) {
  res = res + s
  s = s + 1
}
res
```

```
## [1] 5050
```

Question 8

(5 points)

Notes 2B (2-3)

Repeat Q7, but use a `for()` loop instead.

```
res= 0
for (ii in 1:100) {
  res = res + ii
}
res
```

```
## [1] 5050
```

Question 9

(5 points)

Notes 2B (5,7)

Write a `while()` loop that samples one value from a standard normal (see `rnorm()`), then breaks when the value is greater than 4. Also include an incrementing variable that increments by one with each loop, and display its value when the loop is broken. (In other words: display how many loops occurred before a sampled value of > 4 was observed.) (Note: you can model this with a geometric distribution. The expected number of loops is 1/p = 1/(1-pnorm(4)).)

```
set.seed(1120)
counter= 0
sample= rnorm(1)
while(sample <= 4){
  counter= counter + 1
  sample= rnorm(1)
}
counter
```

```
## [1] 16925
```

Functions

Be sure to refer to Notes 2C, Slide 15, for a link to Google's `R` coding style guide. (Which is now just another Hadley Wickham production.)

As defined in Wikipedia, the logistic function is:

$$f(x) = \frac{L}{1 + e^{-k(x-x_o)}} \, ,$$

where L is the curve's maximum value, x_o is the x-value of the curve's midpoint (i.e., where $y = L/2$), and k is a variable that controls how steep the curve is (i.e., how quickly it transitions from 0 to L).

Question 10

(10 points)

Notes 2C (2,5,7,9)

Write a function `logistic()` that has arguments `x`, `L`, `x.o`, and `k`, and returns $f(x)$ as defined above. Display the return value for $x = x_o$ and ensure that it is $L/2$. You need not add any comments describing the function input/output. (Note: in `R`, e^x is `exp(x)`.)

```
logistic= function(x, L, x.o, k) {
  return (L/(1+exp(-k*(x-x.o))))
}
x = 1
x.o = 1
L = 10
k = 2
logistic(x, L, x.o, k)
```

```
## [1] 5
```

Question 11

(5 points)

Notes 2C (2,5,7,9)

Modify your logistic function from Q10 so that `L`, `x.o`, and `k` are assigned reasonable default values. Ensure that you get a return value when you specify `x` alone.

```
logistic= function(x, L=1, x.o=1, k=0) {
  return (L/(1+exp(-k*(x-x.o))))
}
logistic(1)
```

```
## [1] 0.5
```

Question 12

(5 points)

Notes 2C (6)

Do the following three lines of code give the same results? If not, why not?

```
# Uncomment the following lines!
logistic(x=1,L=1,x.o=-1,k=2)
```

```
## [1] 0.9820138
```

```
logistic(k=2,L=1,x=1,x.o=-1)
```

```
## [1] 0.9820138
```

```
logistic(2,1,1,-1)
```

```
## [1] 0.2689414
```

```
The last line gives a different answer
because in the third case
since the argument names are not specified
it takes the default order of x, L, x.o, k
resulting in different values assigned to
the parameters than the values in the first
two cases.
```

Question 13

(5 points)

Notes 2A (5) and 2C (7)

Modify your `logistic()` function again so that it checks to see if `x`, `L`, `x.o`, and `k` are of type "numeric". If any of these variables is not numeric, your function should return the value `NULL` (without quotes). A way to do this for a generic variable `x` would be via a line like

```
# a line of code for type checking
# if ( is.numeric(x) == FALSE ) return(NULL)
logistic= function(x, L=1, x.o=1, k=0) {
  if (is.numeric(c(x, L, x.o, k)) == FALSE){
    return (NULL)
  }else{
    return (L/(1+exp(-k*(x-x.o))))
  }
}
logistic("hi")
```

```
## NULL
```

Using logical operators, you can combine all the checks into a single `if` statement. Verify that `NULL` is returned when at least one of the variables is not numeric.

```
logistic= function(x, L=1, x.o=1, k=0) {
  if (is.numeric(x)== FALSE || is.numeric(L)==FALSE ||is.numeric(x.o)== FALSE || is.numeric(k)==FALSE){
    return (NULL)

  }else{
    return (L/(1+exp(-k*(x-x.o))))
  }
}
logistic("hi")
```

```
## NULL
```

Question 14

(5 points)

Notes 2C (9)

It is often a good practice to record the values of arguments in function output, so that you can look later and determine what values were used when computing $f(x)$. Amend your logistic function so that it returns the values of x, $f(x)$, L, x_o, and k. Call the function once (with two values of x specified, i.e., with x as a two-element vector) and display your output.

```
logistic= function(x, L=1, x.o=1, k=0) {
  if (is.numeric(c(x, L, x.o, k)) == FALSE){
    return (NULL)
  }else{
    fx= L/(1+exp(-k*(x-x.o)))
    return (list(x=x, fx=fx, L=L, x_o=x.o, k=k))
  }
}
logistic(c(1,2), 10, 1, 2)
```

```
## $x
## [1] 1 2
##
## $fx
## [1] 5.000000 8.807971
##
## $L
## [1] 10
##
## $x_o
## [1] 1
##
## $k
## [1] 2
```

Functions: Silly User Tricks

Question 15

(5 points)

Not Covered in Notes

Did you know that a function can return a function? Run the code below and explain what is happening here.

```
h = function(x)
{
  return(function(y){y^x})
}
z = h(2)
z(3)
```

```
## [1] 9
```

```
z = h(3)
z(2)
```

```
## [1] 8
```

```
Function h has one argument x which is the
exponent in the power function call that takes
in y. So in the first definition of z, z is the
square function. In the second definition of z,
z is the cube function.
```

Question 16

(5 points)

Not Covered in Notes

Using the code above as a template, define a function `z()` that subtracts a previously set constant value from the input value. Demonstrate the use of `z()` with two different values for the constant.

```
h= function(x){
  return (function(y){y-x})
}
z= h(5)
z(10)
```

```
## [1] 5
```

```
z= h(36)
z(10)
```

```
## [1] -26
```

Functions: Environments

Question 17

(5 points)

Notes 2C (12-13)

What does the following code output? Explain how that output is generated.

```r
x = 1
h = function() {
  y = 2
  i = function() {
    z = 3
    c(x, y, z)
  }
  i()
}
h()
```

```
## [1] 1 2 3
```

```
Within function h, variable y and function i
are defined. Function i returns a vector of
(x,y,3) since x and y are not defined within
function i, it looks for it in the outer function
finding y to be 2. Similar process for x and see
that it is defined globally as 1. Function h returns
the value from function i, which is the vector
(1, 2, 3).
```

Functions: Recursion

Question 18

(10 points)

Notes 2C (14)

Code what is possibly the simplest example of recursive function use: the factorial function, i.e., $x! = x \times (x-1) \times \cdots \times 1$, for positive whole number x. Note that the name `factorial()` is already used for a function in `R`'s base package, so use `my_factorial()` instead. Comments are not necessary, and note that your function body only need be two lines long. Compute 5!.

```r
my_factorial= function(n){
  ifelse(n==1, n, n*my_factorial(n-1))
}
my_factorial(5)
```

```
## [1] 120
```