Language Research Project

William Keilsohn

May 9, 2017

1. What programming language did you pick?

For this assignment, I choose the programming language R. For the sake of this assignment, I will be including R Studio in my discussion, even though it is not the language its self, but rather is a IED used frequently to write and read R.

1. Why did you choose this programming language?

I have chosen this programming language because it is frequently used in both of my fields (marine science & entomology) to process large data sets and perform statistical analysis. I used it briefly for my senior thesis, but was heavily reliant on help from a graduate student. Now that I will be graduating shortly, I would very much like to be able to do the work relating to this on my own.

1. What basic data types are provided by the language? Examples.

R has the basic data types of integers, numeric/numbers, complex numbers, logical, and characters. Some more complex data types that are frequently considered "basic" are factors, vectors, assignment operators, and data frames. Numeric values are those which are real numbers. This includes whole and decimal values. Integer values are a subset of numeric values that normally involve a letter and number combination. Complex numbers, as their name implies, are numbers involving the complex value i. Logical values are essentially Boolean, and characters are essentially strings. In terms of more complex data sets, vectors are the only one sometimes considered both atomic and combinations of atomic values. Vectors are constructed by concatenating other more primitive values in much the same way a list is made in Dr. Racket.

X <- 1  
class(X) #Numeric Value

## [1] "numeric"

y <- 8L  
class(y) #Integer Value

## [1] "integer"

z <- X \* 1i  
class(z) #Complex Value

## [1] "complex"

a <- 8 > 7  
class(a) #Logical Value

## [1] "logical"

b <- "Science"  
class(b) #Character Value

## [1] "character"

v <-c(X, y, z, a, b) #Vector  
v

## [1] "1" "8" "0+1i" "TRUE" "Science"

1. What operators are provided for the basic data types? Examples.

R provides the basic operators of addition, subtraction, multiplication, division, exponentiation, modulus, and integer division. It also includes logical operators such as less/greater than, less/greater than or equal to, not equal to, "not", "or", "and" and test if X is true. There is also a paste function for strings that works like append, and a concatenate function for vectors that allows you to combine lists, but as the language focuses around data analysis, most of the given operators revolve around numeric and other similar numbers.

t.vec <-c(1:20)  
d.vec <-c(21, 36, 2, 54, 75, 83, 82, 44, 91, 11, 29, 39, 41, 88, 52, 64, 86, 55, 7, 30)  
  
8 \* 9 # 72

## [1] 72

8 + 9 # 17

## [1] 17

10 - 4 # 6

## [1] 6

12 / 3 # 4

## [1] 4

12 %% 6 # 0

## [1] 0

4 ^ 4 # 256

## [1] 256

16 > 12 # True

## [1] TRUE

16 < 12 # False

## [1] FALSE

8 >= 8 # True

## [1] TRUE

8 == 8 # True

## [1] TRUE

7 <= 6 # False

## [1] FALSE

74 != 8 # True

## [1] TRUE

t.vec > 10 & t.vec < 15 # A vector indicating all values between 10 and 15 as True

## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  
## [12] TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE

t.vec < 10 | t.vec > 15 # A vector indicating all values between 10 and 15 as False

## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE  
## [12] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE

isTRUE(length(d.vec) == 20) # False

## [1] TRUE

1. What support is provided for conditionals? Examples.

R relies primarily on if/else statements and their use both nested and not nested. Conditionals also typically rely on both items being compared at the time of the condition's evaluation. This means that one is typically not able to write a blanket statement/function with the intention of inserting a value you wish to compare later. Rather the conditional tests only pre-existing variables/values against each other when performing its evaluation. There are exceptions to this, such as when defining some functions, but this is rarely the case.

Twilight\_Sparkle <- 105  
Pinkie\_Pie <- 65  
  
if (Twilight\_Sparkle > Pinkie\_Pie){  
 print("Twilight is best pony!")  
 } else   
 print("Pinkie in best pony!")

## [1] "Twilight is best pony!"

if (X > 2) {  
 print("The time has come")  
 } else if (y > 8) {  
 print("There is much to do, boy")  
 } else   
 print("There is plenty of time")

## [1] "There is plenty of time"

1. How are functions written? Examples.

In R functions are typically written as using the arrow symbol, i.e. "<-", and an indication that this is either a function of a conditional. This allows one to define a function in terms of a set of arguments. To then call said function one typically writes the function with the arguments they wish for it to act on within parentheses at the end of the function call.

burn.it <- function(x) {  
burn <- x / x  
return (burn)  
}  
burn.it(15) # The answer is 1

## [1] 1

code.it <- function(x) {  
code <-c(x, 0, 1, 1, 0, 0, 0, 1)  
return (code)  
}  
code.it(1) #Returns a vector of eight zeros as ones as a joke.

## [1] 1 0 1 1 0 0 0 1

1. What support is provided for compound data? Examples.

R handles compound data mostly in the form of data frames, arrays, and matrices. It also does handle lists. For data frames R essentially builds what can be thought of as spread sheets to deal with the data. For arrays, and matrices it essentially does the same thing by creating a spread sheet with all the values stored in rows. For arrays, it creates what is essentially a stack of spread sheets or what can be thought of as multiple matrices "stacked" on top of each other. These are used in certain statistical processes and thus are good for certain research related calculations. Lists are much simpler and are stored similar to a vector or how a list would be stored in Dr. Racket. Data from all of these can be called at one's discretion.

D.frame <-data.frame(t.vec, d.vec)  
D.frame

## t.vec d.vec  
## 1 1 21  
## 2 2 36  
## 3 3 2  
## 4 4 54  
## 5 5 75  
## 6 6 83  
## 7 7 82  
## 8 8 44  
## 9 9 91  
## 10 10 11  
## 11 11 29  
## 12 12 39  
## 13 13 41  
## 14 14 88  
## 15 15 52  
## 16 16 64  
## 17 17 86  
## 18 18 55  
## 19 19 7  
## 20 20 30

D.matrix <-matrix(t.vec, d.vec, ncol = 2, nrow = 20)  
D.matrix

## [,1] [,2]  
## [1,] 1 2  
## [2,] 3 4  
## [3,] 5 6  
## [4,] 7 8  
## [5,] 9 10  
## [6,] 11 12  
## [7,] 13 14  
## [8,] 15 16  
## [9,] 17 18  
## [10,] 19 20  
## [11,] 1 2  
## [12,] 3 4  
## [13,] 5 6  
## [14,] 7 8  
## [15,] 9 10  
## [16,] 11 12  
## [17,] 13 14  
## [18,] 15 16  
## [19,] 17 18  
## [20,] 19 20

D.array <-array(1:27, c(3, 3, 3))  
D.array

## , , 1  
##   
## [,1] [,2] [,3]  
## [1,] 1 4 7  
## [2,] 2 5 8  
## [3,] 3 6 9  
##   
## , , 2  
##   
## [,1] [,2] [,3]  
## [1,] 10 13 16  
## [2,] 11 14 17  
## [3,] 12 15 18  
##   
## , , 3  
##   
## [,1] [,2] [,3]  
## [1,] 19 22 25  
## [2,] 20 23 26  
## [3,] 21 24 27

W <-list(d.vec, t.vec)  
W

## [[1]]  
## [1] 21 36 2 54 75 83 82 44 91 11 29 39 41 88 52 64 86 55 7 30  
##   
## [[2]]  
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

1. What support is provided for arbitrary sized data? Examples.

Arbitrary size data is typically handled in R via the creation of blank data frames or the like which can later be filled in with the appropriate information. As R is typically used to handle data that is either already collected or which the researcher has an idea of how many tests/trial they are going to run, this is rather in frequent in terms of actual use or practice. Thus R doesn't really have an advanced or complex way of dealing with data of this form. Rather it asks the user for an estimate and goes from there.

Main\_Six <-data.frame(matrix(ncol=3, nrow=6))  
Main\_Six

## X1 X2 X3  
## 1 NA NA NA  
## 2 NA NA NA  
## 3 NA NA NA  
## 4 NA NA NA  
## 5 NA NA NA  
## 6 NA NA NA

b.col <-c("Name", "Color", "Element")  
b.row <-c("Pony 1", "Pony 2", "Pony 3","Pony 4", "Pony 5", "Pony 6")  
colnames(Main\_Six) <- b.col  
row.names(Main\_Six) <- b.row  
Main\_Six

## Name Color Element  
## Pony 1 NA NA NA  
## Pony 2 NA NA NA  
## Pony 3 NA NA NA  
## Pony 4 NA NA NA  
## Pony 5 NA NA NA  
## Pony 6 NA NA NA

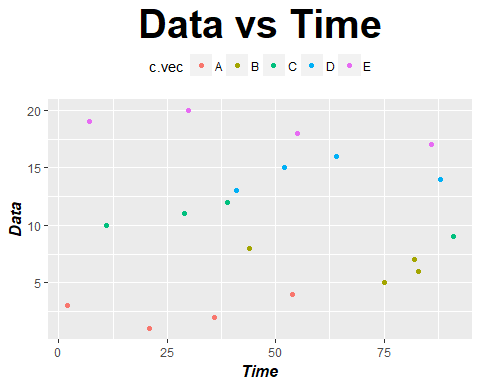
names.vec <-c("Twilight Sparkle", "Apple Jack", "Pinkie Pie", "Rarity", "Rainbow Dash", "Fluttershy")  
color.vec <-c("Purple", "Orangle", "Pink", "White", "Blue", "Yellow")  
element.vec <-c("Magic", "Honesty", "Laughter", "Generosity", "Loyalty", "Kindness")  
Main\_Six[,1] <-names.vec  
Main\_Six[,2] <-color.vec  
Main\_Six[,3] <-element.vec  
Main\_Six

## Name Color Element  
## Pony 1 Twilight Sparkle Purple Magic  
## Pony 2 Apple Jack Orangle Honesty  
## Pony 3 Pinkie Pie Pink Laughter  
## Pony 4 Rarity White Generosity  
## Pony 5 Rainbow Dash Blue Loyalty  
## Pony 6 Fluttershy Yellow Kindness

1. What are the strengths and weaknesses of the language? Examples.

The greatest strengths of this language is its ability to run advanced statistical models and to to then graph the resulting relationships of those models. It is also has the unique advantage of importing data from other sources and allowing one to perform all computations or analysis of that data in one convenient place. As well, it is free, widely distributed, and easily crosses platforms allowing for researchers to share information and data easily. The biggest weakness unfortunately is that it really only works on data. Any thing more advanced, such as image rendering images, designing programs to run outside of the IED, or more complex purposes, just doesn't work so well. It's purpose really is data analysis.

library(ggplot2)  
c.vec <-c("A", "A", "A", "A","B", "B", "B", "B", "C", "C", "C", "C", "D", "D", "D", "D", "E", "E", "E", "E")  
D.frame2 <- data.frame(t.vec, d.vec, c.vec)  
par(mar=c(1,1,1,1))  
p <- qplot(d.vec, t.vec, data=D.frame2, color=c.vec,   
 xlab="Time", ylab="Data", main="Data vs Time")  
p + theme(axis.title=element\_text(face="bold.italic",  
 size="12",   
 color="black"),   
 legend.position="top",  
 plot.title = element\_text(face="bold",  
 size = "30", color = "black",  
 hjust = 0.5))



model1 <-aov(d.vec ~ c.vec + t.vec, data=D.frame2)  
summary(model1)

## Df Sum Sq Mean Sq F value Pr(>F)  
## c.vec 4 4504 1125.9 1.667 0.213  
## t.vec 1 1225 1225.0 1.814 0.199  
## Residuals 14 9457 675.5