Computer Programming

Foundations for Understanding and Writing Quality Code in R

Investment Management Division

Arizona State Retirement System

March 22, 2017

Outline

1. Computer Programming

2. R Syntax

Outline

1. Computer Programming

2. R Syntax

Why write computer programs?

- A computer program is one or more lines of “code” providing instructions to a computer to do something you want to do
- We write programs for a number reasons, among them
  - to save time
    - If tasks are repetitive it can save time and money to automate them
  - to take advantage of sophisticated methods that are impractical to do otherwise
    - quantitative methods involving large data sets have revolutionized many fields, including finance
  - for transparency and reproducibility
    - reproducible research presents code along with data allowing readers to form their own judgment on the validity of the work
    - transparent coding improves quality by exposing and reducing model error

What makes a program good?

- It does something useful
- It does it correctly
  - with transparency about methods and assumptions
  - ideally with open source code allowing for verification
- It is usable
  - with documentation and workable interfaces suitable to the context
  - with clearly defined "data contracts" defining
    - requirements for inputs stating what can and can't be processed and how it needs to be formatted
    - data validation and clear error messages for data that don't meet the requirements
    - clear definition for the form and content of the data returned by the process
- It is intuitive
  - the architecture contemplates how the data arrives and how the outputs will be used
  - doesn't need (a lot of) pre or post processing

A Sample Program

- Let's write a program to convert between fahrenheit and celsius
- Recall
  - $F = 32 + 1.8 * C$
  - $C = \frac{F-32}{1.8}$
- A function that does this is on the next page

```r
temp_convert=function(degree,sys) {
  # function to convert between fahrenreit and celsius
  # Arguments:
  # degree -- numeric value of temperature,
  #       can be vector but not higher dimension
  # sys -- system of delivered temperature,
  #       must be F or C
  # Returns:
  # a number or vector of the converted temperature,
  #       with a name(ans) equal to either
  #       "Fahrenheit" or "Celsius"
  # validate inputs
  if (!(sys=="F" | sys=="C")) stop("sys must be F or C")
  if (!is.numeric(degree)) stop("degree must be numeric")
  if (length(dim(degree))>1)
    stop("dimension of degree must be no more than vector")
  # do the calculations
  if(sys=="F") {
    ans=(degree-32)/1.8
    names(ans)=rep("Celsius",length(ans))
  }
  if(sys=="C") {
    ans=32+1.8*degree
    names(ans)=rep("Fahrenheit",length(ans))
  }
  return(ans) # return the answer
}
```

Example use of function

- Note the function meets the requirements for being “good” including
  - documentation describing how to use it
  - data validation with OK error messages

```
temp_convert(32,"F")

## Celsius
##       0

temp_convert(32,"D")

## Error in temp_convert(32, "D"):  sys must be F or C

temp_convert(c(-10,0,10,20,30,40),"C")

## Fahrenheit Fahrenheit Fahrenheit Fahrenheit Fahrenheit Fahrenheit
##         14         32         50         68         86        104
```

- 0 Celsius is the same as 32 Fahrenheit

Outline

1 Computer Programming

2 R Syntax

Special Characters

- “=” assigns a value to an object
- “+”, “-”, “/”, “*”, “^”, “.” are the standard arithmetic operators
- “:” generates a sequence; 1:4 is 1,2,3,4
- logic operators
  - “&” is and
  - “|” is or
  - “!” is not
  - “==” for logical equals (different from assignment)
  - “<”, “<=”, “>” and “>=” are less than, less than or equal, greater than, greater than or equal
- parenthesis
  - act as delimiters for the beginning and end of arguments to a function
  - also act to control order of calculation
  - both of these work the same as what you may be used to with excel
- “,” separates arguments to a function
- “[“ and “]” (brackets)
  - used to index in to vectors, data frames or arrays
  - double brackets “[[“ and “]]” index in to lists
- “{” and “}”
  - delimiters for code chunks
  - used in function definitions, if/else constructs, etc
- “%” is used for in line functions like “%in%” or “%*%”
- “$” grabs a column from a data frame (or an element of a list) by name as in my_dataframe$col_name
- “#” indicates the beginning of a “comment” in code which should be explanatory and is not executed as R code
- “~” is function notation, as in lm (y ~ x)

Assign to Create Objects

```
k=3 #put a number in a variable
ages=c(4,6,7) #create a vector of numbers for ages of children
kid_names=c("Bill","Karen","Tom")  #create a vector of names
kids=data.frame(kid_names,ages) #create a data frame of the kids
kids

##   kid_names ages
## 1      Bill    4
## 2     Karen    6
## 3       Tom    7

temperature=c(98.6,100.9,104)
location=c("School","Home Sick","Emergency Room")
kids=cbind(kids,temperature, location) #add temperature and location to the data frame
kids

##   kid_names ages temperature       location
## 1      Bill    4        98.6         School
## 2     Karen    6       100.9      Home Sick
## 3       Tom    7       104.0 Emergency Room
```

Indexing and Subsetting

```
kid_names[2]  #the second child

## [1] "Karen"

kid_names[ages==max(ages)] #the oldest child

## [1] "Tom"

kids$ages #the ages of the children

## [1] 4 6 7

kids[,2] #the ages of the children

## [1] 4 6 7

kids[3,] #information about the third child

##   kid_names ages temperature       location
## 3       Tom    7         104 Emergency Room
```

More Indexing and Subsetting

```
kids[2,3] #for the second child (row 2), what is the temperature (column 3)

## [1] 100.9

kids[kids$kid_names=="Karen","temperature"] #Karen's temperature

## [1] 100.9

subset(kids,kids$temperature>98.6) #which kids are sick

##   kid_names ages temperature       location
## 2     Karen    6       100.9      Home Sick
## 3       Tom    7       104.0 Emergency Room
```

Nesting

```
pointers=c(3,2,4,6)
data=c(12,14,16,18,20,22,24,26)
data[pointers[2]] #value of the data pointed at by the second pointer

## [1] 14

data[sum(1,2)]

## [1] 16

cumsum(1:4)[3]

## [1] 6

cumsum((1:4)[3])

## [1] 3
```

Pipes (optional)

Nesting too deeply makes code hard to read. For example:

```
floor((2+4)*(2+log(sqrt(3*(2+exp(4)))/12)))

## [1] 12
```

When this happens, break your code in to multiple lines and store intermediate results. Alternatively, use the magrittr[1] package which implements a “pipe” coding construct in R. The same calculation as above implemented with this technique:

```
require(magrittr)

## Loading required package:  magrittr

exp(4) %>%
  add(2) %>%
  multiply_by(3) %>%
  sqrt() %>%
  divide_by(12) %>%
  log() %>%
  add(2) %>%
  multiply_by(2+4) %>%
  floor()

## [1] 12
```

[1] https://en.wikipedia.org/wiki/The_Treachery_of_Images

Homework

- Write a function to calculate the monthly payment on a mortgage
- The formula is as follows

$$payment = balance * monthly.rate * \frac{1}{1-(\frac{1}{1+monthly.rate})^n}$$

$$monthly.rate = annual.rate/12$$

$$n = years.in.mortgage * 12$$

- So, the arguments to the function are the balance, the stated annual rate and the number of years
- The function returns the payment
- Read the files loanamort.html and loanamort.rmd in the rug7 folder for derivations and a fancy version of this function