R Best Practices

Richard Gayle

Carpentries Workshop, R Programming, GFZ Potsdam

November 22nd and 23rd, 2021

Comment as You Go

What does this code do?

```
mod<-function(a,n){
        return(a-floor(a/n)*n)
                                }
```

Comment as You Go

What does this code do?

```
mod<-function(a,n){
        return(a-floor(a/n)*n)
                                }
```

Things are clarified by a helpful comment introduced by #:

```
# This defines a function 'mod' which
# gives the remainder of integer a
# when divided by positive integer n.

mod<-function(a,n){
        return(a-floor(a/n)*n)
                                }
```

Load Libraries Upfront

It's helpful to load packages at the beginning of a script.

Load Libraries Upfront

It's helpful to load packages at the beginning of a script. That way potential users are forewarned about what they might need to install.

Load Libraries Upfront

It's helpful to load packages at the beginning of a script. That way potential users are forewarned about what they might need to install.

```
library(ggplot2)
library(reticulate)
library(rhandsontable)
```

Avoid 'Hardloading'

Rather than:

```
my_data<-read.csv('my_data.csv')
thingy<-somefunction(my_data)
write(thingy, 'my_fixed_data.csv')
```

Avoid 'Hardloading'

Rather than:

```
my_data<-read.csv('my_data.csv')
thingy<-somefunction(my_data)
write(thingy, 'my_fixed_data.csv')
```

try

```
inputfile_name<-'my_data.csv'
outputfile_name<-my_fixed_data.csv'
my_data<-read.csv(file_name)
thingy<-somefunction(my_data)
write(thingy, outputfile_name)
```

Use Care with setwd()

Use

`getwd()`

to find the current working directory.

Use Care with setwd()

Use

```
getwd()
```

to find the current working directory.

```
setwd('path_name')
```

sets the current working directory.

Use Care with setwd()

Use

```
getwd()
```

to find the current working directory.

```
setwd('path_name')
```

sets the current working directory.
Use it with caution in scripts as you might wish to the script you write in a variety of different directory structures.

Use Care with setwd()

Use

```
getwd()
```

to find the current working directory.

```
setwd('path_name')
```

sets the current working directory.
Use it with caution in scripts as you might wish to the script you write in a variety of different directory structures.
If you use it in a script, try to do so at the outset to give users a heads-up.

Modularize Your Code, I

This is tough to follow:

```
mod<-function(a,n){
        return(a-floor(a/n)*n)
                                    }
gcd<-function(a,b){
         if (a==0 | b==0)
               {return(max(abs(a), abs(b)))}
         else {
             a<-abs(a)
             b<-abs(b)
             while (b>0){
                   r<-mod(a,b)
                   a<-b
                   b<-r}
                 }
          return(a)}
```

Modularize Your Code, II - Better Version

```
# The function 'mod' finds the residue class of
# integer a modulo positive integer n.

mod<-function(a,n){
        return(a-floor(a/n)*n)
                                        }
```

And then ...

Modularize Your Code, III - Better

```
# The function 'gcd' uses the 'mod' function
# and the Euclidean Algorithm to find the
# gcd of integers a and b.

gcd<-function(a,b){
        if (a==0 | b==0)
            {return(max(abs(a), abs(b)))}
        else {
            a<-abs(a)
            b<-abs(b)
            while (b>0){
                r<-mod(a,b)
                a<-b
                b<-r}
              }
         return(a)}
```

Use Multiple Scripts

If your project requires multiple functions, divide these up into multiple scripts organized by theme.

Use Multiple Scripts

If your project requires multiple functions, divide these up into multiple scripts organized by theme. For example, in writing R Shiny apps recently, I create quite a few special graphics functions and special 'mathy' functions.

Use Multiple Scripts

If your project requires multiple functions, divide these up into multiple scripts organized by theme. For example, in writing R Shiny apps recently, I create quite a few special graphics functions and special 'mathy' functions. I divide these into seperate scripts and then load these using the 'source' command:

Use Multiple Scripts

If your project requires multiple functions, divide these up into multiple scripts organized by theme. For example, in writing R Shiny apps recently, I create quite a few special graphics functions and special 'mathy' functions. I divide these into seperate scripts and then load these using the 'source' command:

```
source('graphfncs.r')
source('mathfncs.r')
```

Use Multiple Scripts

If your project requires multiple functions, divide these up into multiple scripts organized by theme. For example, in writing R Shiny apps recently, I create quite a few special graphics functions and special 'mathy' functions. I divide these into seperate scripts and then load these using the 'source' command:

```
source('graphfncs.r')
source('mathfncs.r')
```

Make certain that the source files are in the current working directory.

C'Mon, Man, Keep it Together!

It's also useful to keep all the components of a project in the same directory or collection of directories.

C'Mon, Man, Keep it Together!

It's also useful to keep all the components of a project in the same directory or collection of directories.
Then you can use shorter, relative paths in calling source files or loading data.

C'Mon, Man, Keep it Together!

It's also useful to keep all the components of a project in the same directory or collection of directories.
Then you can use shorter, relative paths in calling source files or loading data.

```
source(\graphics\'elliptic.r')
```

as opposed to

C'Mon, Man, Keep it Together!

It's also useful to keep all the components of a project in the same directory or collection of directories.
Then you can use shorter, relative paths in calling source files or loading data.

```
source(\graphics\'elliptic.r')
```

as opposed to

```
source('\Users\rcg6824\Docs\math\Shiny\EICrvs\
        graphics\'elliptic.r')
```

Odds

If you are using R from the command line or in R Studio it's helpful to begin with a clean slate, so to speak.

Odds

If you are using R from the command line or in R Studio it's helpful to begin with a clean slate, so to speak. That is to say, it's confusing to begin by automatically loading a saved workspace which might contain a variety of objects which you created in an earlier session the meanings and uses of which you've completely forgotten.

Odds

If you are using R from the command line or in R Studio it's helpful to begin with a clean slate, so to speak. That is to say, it's confusing to begin by automatically loading a saved workspace which might contain a variety of objects which you created in an earlier session the meanings and uses of which you've completely forgotten.
If you would like to save important information, the command from the package 'devtools'

```
S<-sessionInfo()
```

will save important infiormation about current session, most particularly packages loaded.

Odds

If you are using R from the command line or in R Studio it's helpful to begin with a clean slate, so to speak. That is to say, it's confusing to begin by automatically loading a saved workspace which might contain a variety of objects which you created in an earlier session the meanings and uses of which you've completely forgotten.
If you would like to save important information, the command from the package 'devtools'

```
S<-sessionInfo()
```

will save important infiormation about current session, most particularly packages loaded.

And Ends

`ls()`

lists all the objects defined in the current session.

And Ends

`ls()`

lists all the objects defined in the current session.

`rm(list=ls())`

expunges all these objects and, consequently, should be used with caution.