2. Programming for data analysis

Martin Johnsson
IFM Biology

Languages one may encounter

A silly example

We have a list of numbers stored on file:

101
22
23
1001
…

We want the sum.

R

```
numbers <- read.table("numbers.txt")
sum(numbers)
```

Python

```
import numpy as np
numbers = np.loadtxt("numbers.txt")
print(np.sum(numbers))
```

Perl

```
use strict;
use warnings;

open(my $file, "numbers.txt");

my $sum = 0;
while (my $number = <$file>) {
 $sum = $sum + $number;
}
print($sum, "\n");
```

Running scripts from command line

```
python sum.py
perl sum.pl
R CMD BATCH sum.R
```

C++

```
#include <iostream>
#include <fstream>
using namespace std;

int main (int argc, char *argv[]) {
  double sum = 0;
  double number = 0;
  ifstream numbers_file (argv[1]);

  while (numbers_file >> number) {
    sum += number;
  }
  cout << sum << endl;
}
```

C++

Compiling – translating from human to machine:

```
g++ -o sum sum.cpp
```

Running (on unix-like operating systems):

```
./sum numbers.txt
```

Important concepts

Input/output

Data structures

Functions

Control flow

R: Input

- read.table family of functions (read.table, read.delim, read.csv, read.csv2)
- readr package (read_csv etc) used by *Import dataset* button in later RStudio versions

```
data <- read.table("data.csv", header = TRUE,
                   sep = "\t", dec = ".",
                   skip = 2, na.strings = "-",
                   strip.white = TRUE,
                   stringsAsFactors = FALSE)
```

R: output

- write.table, write.csv – text files

```
write.table(my_data,
            file = "my_file.txt", sep = "\t")
```

- save & load – binary R data files

```
save(my_data,
     file = "my_data.Rdata")
load("my_data.Rdata")
```

R: Data structures

- vectors
- data frames / tibbles
- matrices
- lists

R: Vectors

- numeric
- character
- logical
- Often columns of a data frame.

```
some_numbers <- numeric(10)
specific_numbers <- c(1, 2, 4)
mode(specific_numbers)
length(specific_numbers)
```

R: Data frames

The most important tabular data structure.

What you get from read.table & co

```
some_data <- data.frame(id = c("a", "b", "c"),
                        values = c(23.0, 34.1, 14.5),
                        stringsAsFactors = FALSE)
some_data$id
some_data[, "id"]
some_data[c(1, 3), ]

my_data <- read.csv("my_data.csv", stringsAsFactors = FALSE)
```

R: tibbles

“**tibble: Simple Data Frames**
Provides a 'tbl_df' class that offers better checking and printing capabilities than traditional data frames.“

“Tibbles are data.frames with nicer behavior around printing, subsetting, and factor handling.”

Default in later RStudio versions.

R: Matrices

- Contain only one kind of thing, typically numbers.
- Matrix algebra.
- A mathy data structure.

```
some_matrix <- matrix(1:20, ncol = 5, nrow = 4)
t(some_matrix)

identity_matrix <- diag(4)
identity_matrix %*% some_matrix
```

R: Functions

- A mathematical function: $f(x) = x^2 + 1$
- Expressed in R:

```
f <- function(x) {
  x^2 + 1
}
```

R: Lists

- A vector that can store anything, even another list.
- Useful for making your own complex objects.
- Get elements out with double brackets [[ ]]

```
my_list <- list(1, "ponies", lm(y ~ x, my_data))
my_list[[3]]
list(some_number = 1,
     important_message = "ponies",
     model = lm(y ~ x, my_data))
```

R: Functions

```
my_function <- function (argument) {
   ## do something with argument
   ## return results
}
```

- Encapsulates a part of the code
- Makes it general and repeatable
- Last expression will be returned
- Custom functions are just like built-in ones

Control flow

- Functions – repeated application
- Repetition – loops
- Conditions – if, else, and ifelse

```
## for a single number
if (a > 10) {
  a_capped <- 10
} else {
  a_capped <- a
}

## for a whole vector at a time
a_capped <- ifelse(a > 10, 10, a)
```

Three useful tricks

- Long form vs wide form

```
library(reshape2)
melted <- melt(unicorns, id.vars = c("id", "colour"))
```

Three useful tricks

- Applying a function to a subset of a data frame

```
library(plyr)
unicorn_summary <- ddply(unicorns,
                         c("diet, "colour"),
                         my_function)
```

Three useful tricks

- Finding and replacing text with stringr

```
library(stringr)
text <- c("something", "nothing”,
          "anything", "nil", "void")
str_match(text, pattern = "th")
```

Exercise 2

Homework 2: The unicorn expression dataset