Programming Fundamentals

- Programming concepts and understanding of the essentials of programming languages form the basis of computing.

Goals

- To outline what aspects of a language (specifically R) you might want to teach your students
- Discuss how to teach this effectively
  concepts, examples, motivation, context
- Discuss aspects of the language that not all users are familiar with
  and to show the logic/simplicity of the language

- Three aspects to statistical programming
  - interactive use for exploratory data analysis
  - programming numerically intensive tasks
  - writing functions/software for reuse
- Obviously related, and there is a natural progression.

- Some students won't have seen a "programming language" before
- Some may have seen HTML - a declarative language that has no control flow.
- Many will be unfamiliar with the command-line REPL - Read-Eval-Print-Loop
- Explain the difference between a spreadsheet and a programming environment.

- For some students, we have to get them past the "why isn't this like X? It's so lame!"
- So need to show them how it is useful, and how the language constructs help them to do things.
- So we have to get them to want to do something which involves, e.g. a lot of repetition.
  e.g. give them a corpus of e-mail messages, 1 per file, or person per file and ask how many messages do we have?
- Get them to consider the language as a servant.

What is a computation?

- Transformation from on or more inputs to an output
- Transition from old state to new state
- Algorithm set of directions for carrying out a computation in terms of other simpler computations
- Examples
  - Find the average annual rainfall at a weather station
  - Crop a digital photo
  - Sort the mail by sender in your mail program

Language Model

- Critical to teach the concepts, structure, logic and design of the language
- Students will often gravitate to "how to" and mimicking rather than abstracting the specific to gain this "higher understanding" of the language.
- Syntax versus Semantics and computational model.
- Getting past details allows them to
  - reason about problems
  - apply their knowledge of one language to another.

- Most of the languages we will use are
  - interpreted
  - high-level
  - garbage collected
  - "interactive"
  - & have large libraries
- R, MATLAB, ... - triple of
  - language
  - interpreter
  - environment (packages)

Getting Started

- Get students familiar with the basics of the environment. Do this with an interactive demo. Place online after class a transcript of session (with comments).
- How to start (command line or GUI) and quit q()
- How to use R as a calculator
- Show them the continuation prompt

```
> 2 *
+  3
```

Terminology

- **Invoke** a computation with an **expression**
- Pass the expression off to the computer to **evaluate**
- **Return** a value or output of the expression

Syntax

- Do arithmetic - 1 + pi
- Order of operations - log(100) + sin(pi/2) * 15
  (log(100) + sin(pi/2)) * 15
- Simple plots - curve(sin, -pi, pi)
- Assign results to variables

```
x =  1 + pi
print(x)
x          result of evaluating a non-assignment => print
```

Transcript

```
> 1 + pi
[1] 4.141593

> log(100) + sin(pi/2) * 15 #multiplication first, log base e
[1] 19.60517

> (log(100) + sin(pi/2)) * 15  #parentheses modify order
[1] 84.07755

> (log(100, base = 10) + sin(pi/2)) * 15 #change parameter
  called base
[1] 45
```

Transcript

```
> curve(sin, -pi, pi)   #curve function called, 3 arguments

> x = 1 + pi
> print(x)
[1] 4.141593
> x
[1] 4.141593

> x = rnorm(100)
> mean(x)
[1] 0.02533064
```

- Parsing: break down expression –
  - white space,
  - digits 2x vs 2*x,
  - naming conventions
- No declarations of variables needed or type information

- Explore assignments
  - stored in session area – global environment
  - see names of existing variables – objects()
  - where does curve() come from?
    - find("curve")
    - search() and the concept of the search path
      - how does R use this

```
> pi= 3
> find("pi")
[1] ".GlobalEnv"    "package:base"
```

- Remove one or more objects – rm(x, pi)
- Save variables for future use –
  save(x, z, file ="myvars.rda")
- Restore – load("myfile.rda") (Where are these?)
- Keep track of code – history()

- What about '+' in 1 + 2?
  Does R just know about that?
- find("+")
  "package:base"
- 1 + 2 is actually a function call
  `+`(1, 2)
- In fact, everything in R is a function call
  simple, single concept that makes lots of things easy to reason about, and several computational tasks feasible. (infix vs function style)
- Notion of function call is similar to other languages
  e.g. shell – find . -name '*.R'

Everything's an Object

- If we can see 'x', can we see '+'?
- Print the value of + – poor choice!
  `+`
  sin
- We can pass a function as an argument to a function
  curve(sin)
- So functions are first class values.
- In fact, every value is a first class object.

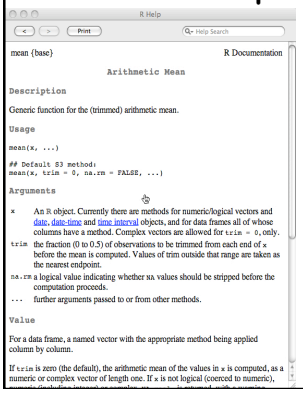
Function Calls

- What's a function?
  - for now, a black box that takes zero or more inputs, and returns a value, an object.
  - leads to 2 issues:
    - how do we specify the inputs
      parameter matching
    - what sort of values can we pass in and get out?
      data structures

Getting Help

- How to get help (help.start)
- It is useful to show them the elements of a help page and how to read them,
  interpret some of the "ambiguous" content,
  explain some of the terms
  follow the See Also
  connect to the manuals which give some more details

Help Page



help(mean)

Description - what the function does

Usage – definition/how to call the function

Arguments - purpose of each argument

Value – return value from function

References -

See Also – related functions

Examples -

Work environment

- Where do you write your code? Not in a Word document
  - Text editors – the GUI's editor, eMacs & ESS
- How do you find your data, code?
  - File system – trees
  - Simple shell commands – cd, ls, pwd, cp, mv, rm, mkdir, rmdir
- Can you use a faster computer? – ssh, scp