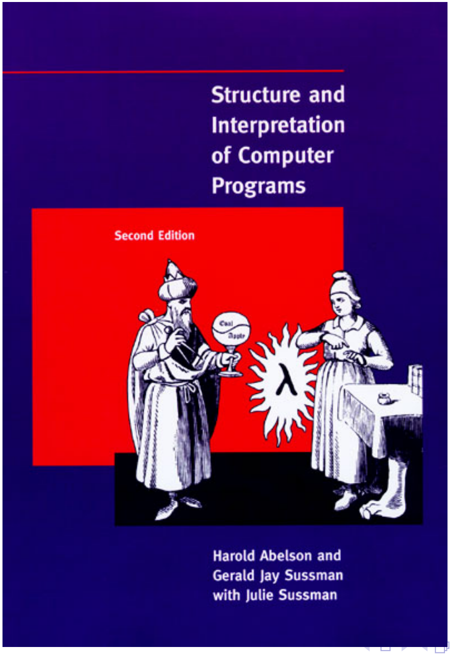
Introduction to Programming: the R perspective

Peter D Smits

May 2, 2013

What is programming?


Structure and Interpretation of Computer Programs
Second Edition
λ
Harold Abelson and Gerald Jay Sussman with Julie Sussman

What is programming?

Structure of Interpretation of Computer Programs by Abelson and Sussman 1996 page 1.

> *We are about the study the idea of a computational process. Computational processes are abstract beings that inhabit computers. As they evolve, processes maniputate other acstract things called data. The evoution of a process is directed by a pattern of rules called a program. People create programs to direct processes. In effect, we conjure the spirits of the computer with our spells.*

Continued. . .

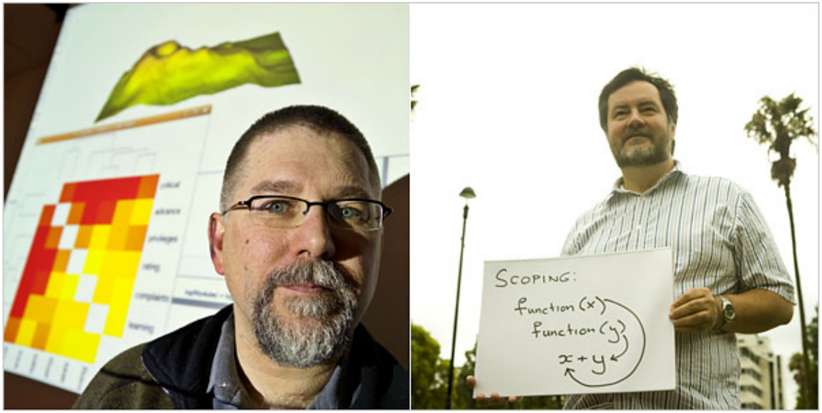
What is programming?

A computation process is indeed much like a sorcerer's idea of a spirit. It cannot be seen or touched. It is not composed of matter at all. However, it is very real. It can perform intellectual work. It can answer questions. It can affect the world by disbursing money at a bank or by controlling a robot arm in a factory. The programs we use to conjure processes are like a sorcerer's spells. They are carefully composed from symbolic epressions in arcane and esoteric programming languages that prescribe the tasks we want our process to perform.

R: a brief history

R is a programming language based on the S language.

Originally written by Ross Ihaka and Robert Gentleman, now by a team.



NyTimes

R: a brief history

Existed in a usable form since approximately 1997.

R is kind of a messy language and pretty slow. Lots of other problems/concerns.

Main advantages of R for statistical analysis

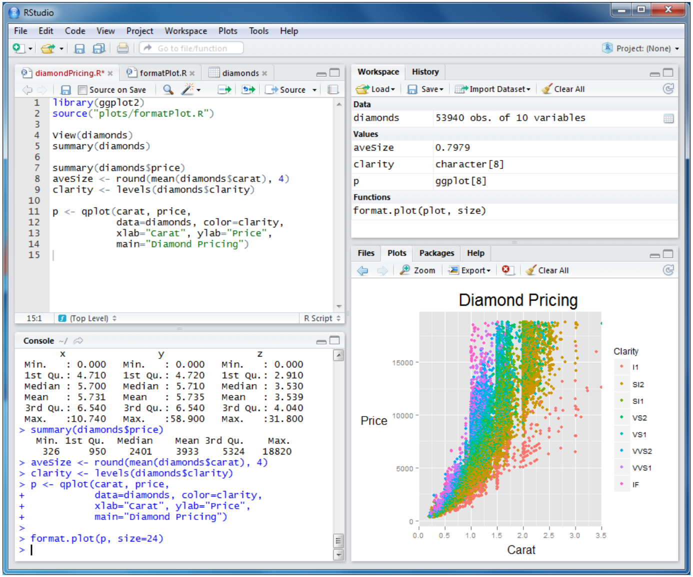
- designed with data in mind
- huge package universe
- lots of online resources
- graphics
- dynamic documents

Console and scripts

The console (or REPL) is the R command line. It is the blinking cursor you stare at when you first open R. It is useful for testing bits of code and hunting for bugs in your code.

Script files are text times with a series of commands/expessions in a row. These are then run to perform analysis. Scripts allow you to edit code and save it for later use. Write scripts, don't just use the console.

RStudio



```r
library(ggplot2)
source("plots/formatPlot.R")

view(diamonds)
summary(diamonds)

summary(diamonds$price)
aveSize <- round(mean(diamonds$carat), 4)
clarity <- levels(diamonds$clarity)

p <- qplot(carat, price,
           data=diamonds, color=clarity,
           xlab="Carat", ylab="Price",
           main="Diamond Pricing")
```

```
Data
diamonds     53940 obs. of 10 variables
Values
aveSize      0.7979
clarity      character[8]
p            ggplot[8]
Functions
format.plot(plot, size)
```

```
       x                y                z        
 Min.   : 0.000   Min.   : 0.000   Min.   : 0.000  
 1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910  
 Median : 5.700   Median : 5.710   Median : 3.530  
 Mean   : 5.731   Mean   : 5.735   Mean   : 3.539  
 3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040  
 Max.   :10.740   Max.   :58.900   Max.   :31.800  
> summary(diamonds$price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    326     950    2401    3933    5324   18820 
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+            data=diamonds, color=clarity,
+            xlab="Carat", ylab="Price",
+            main="Diamond Pricing")
> 
> format.plot(p, size=24)
> 
```

Using the console or REPL

Writing our first script

Write the following code in a script file.

Flow control

Flow control is how you tell the computer what order to do things in.

Use TRUE/FALSE (Boolean) statements.

Basics of conditional statements

Think about what these statements will do then try then in the console.

```
TRUE == TRUE

TRUE == FALSE

FALSE != TRUE

2 > 1

2 < 1

is.numeric(1)
```

if statements

```
if(x = TRUE) {
  print(x)
}

if(x = TRUE) {
  print('yay')
} else {
  print('boo')
}

if(x = TRUE) {
  print('yay')
} else if(x = FALSE) {
  print('what?')
} else {
  print('why?')
}
```

while statements

```
while (y < 100) {
    y <- y + 1
}

while (y < 100) {
    if (y < 0) {
        break
    }
    y <- y + 1
}
```

CAUTION: `while loops can be infinite...`

for statements

```
# i write these a little weird so they are clearer and
# safer

for (ii in seq(10)) {
    print(ii)
}
```

Writing our first function

Using what we know now, let's write our own useful function.

Let's duplicate the sum function.

Writing our first function

First, let's find out what sum does.

```
x <- seq(5)
sum(x)

## [1] 15


y <- c(1, 1)
sum(y)

## [1] 2
```

Writing our first function

```r
sum.prime <- function(x) {
  y <- 0  # storage

  # add every value of x to y
  for(i in seq(length(x))) {
    # seq is for making sequences
    # length determines how long a vector is

    y <- y + x[i]
  }

  # return determines the output of a function
  return(y)
}
```