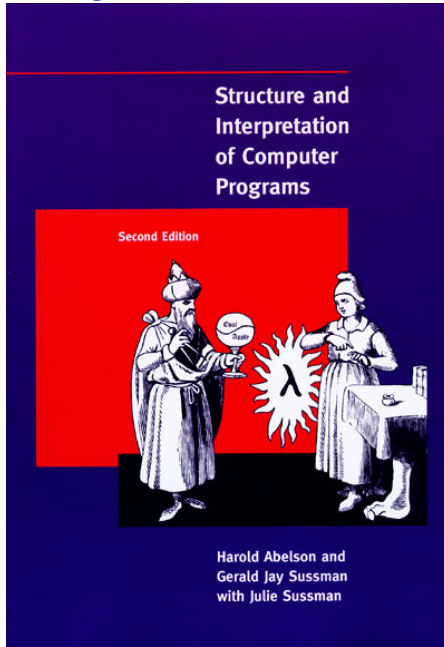


# Introduction to Programming: the R perspective

Peter D Smits

May 9, 2013

# What is programming?



# What is programming?

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

*We are about to study the idea of a computational process. Computational processes are abstract beings that inhabit computers. As they evolve, processes manipulate other abstract things called data. The evolution 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 expressions 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.

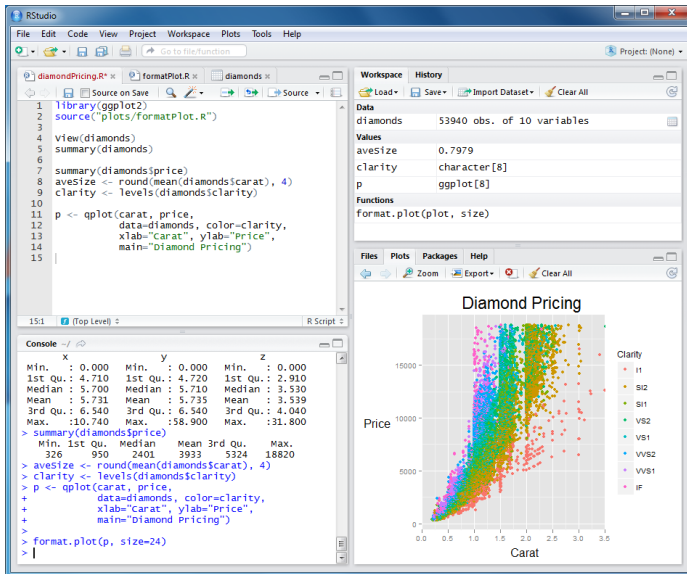
Main advantages of R for statistical analysis

- ▶ 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/expressions 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. These are plain text files with specific extensions. “.r” or “.R” in the case of R.





# Using the console or REPL

Type straight into the console.

```
1  
## [1] 1  
  
x <- 1  
  
x + 1  
## [1] 2  
  
y <- x + x  
  
y  
## [1] 2
```

# Using the console or REPL

You can scroll up/down using the arrow keys.

Useful functions. Give them a try.

```
ls()
```

```
sessionInfo()
```

## Hold on. What did we just do?

```
<-
```

This is called an assignment. “Binds” a value to a variable/object so it can be called later. It can be lots of things, but lets stick with numerics (numbers) and strings (characters). Strings are in quotes.

```
a <- 1
```

```
a
```

```
## [1] 1
```

```
b <- "hello"
```

```
b
```

```
## [1] "hello"
```

Play around with this. Try different variable names. Some aren't valid. You can even bind a variable to a variable like we did earlier.

# Syntax

Syntactic rules are those which govern the order of words. In programming, this means the way we have to write out our statements. The way we assign a variable is a type of syntactic rule.

In R, assignments are flexible which should be concerning.

```
x <- 1  
y = 1  
1 -> z
```

Which one should we use? I like the first. No one likes the third and you should probably never use it. Whatever you chose, stick to it.

# Writing our first script

Write the following code in a script file.

# Comments

Comments are unevaluated text that we can include in a script file. Useful for notes and reminders.

In R, comments are preceded by a `#` symbol. Here is an awful example.

```
x <- 1 # i've made x the value 1
```

Comments are so that other people who are looking at your code can more quickly understand what is going on. Remember, you in 6 months counts as a different person. In CS classes, you get points off for poor commenting.

Comments should explain the why, not the what.

Always comment your code.

# 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 them 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

```
my.sum <- 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)  
}
```

# Next time

## Basic programming concept

- ▶ types
- ▶ classes
- ▶ objects
- ▶ functions