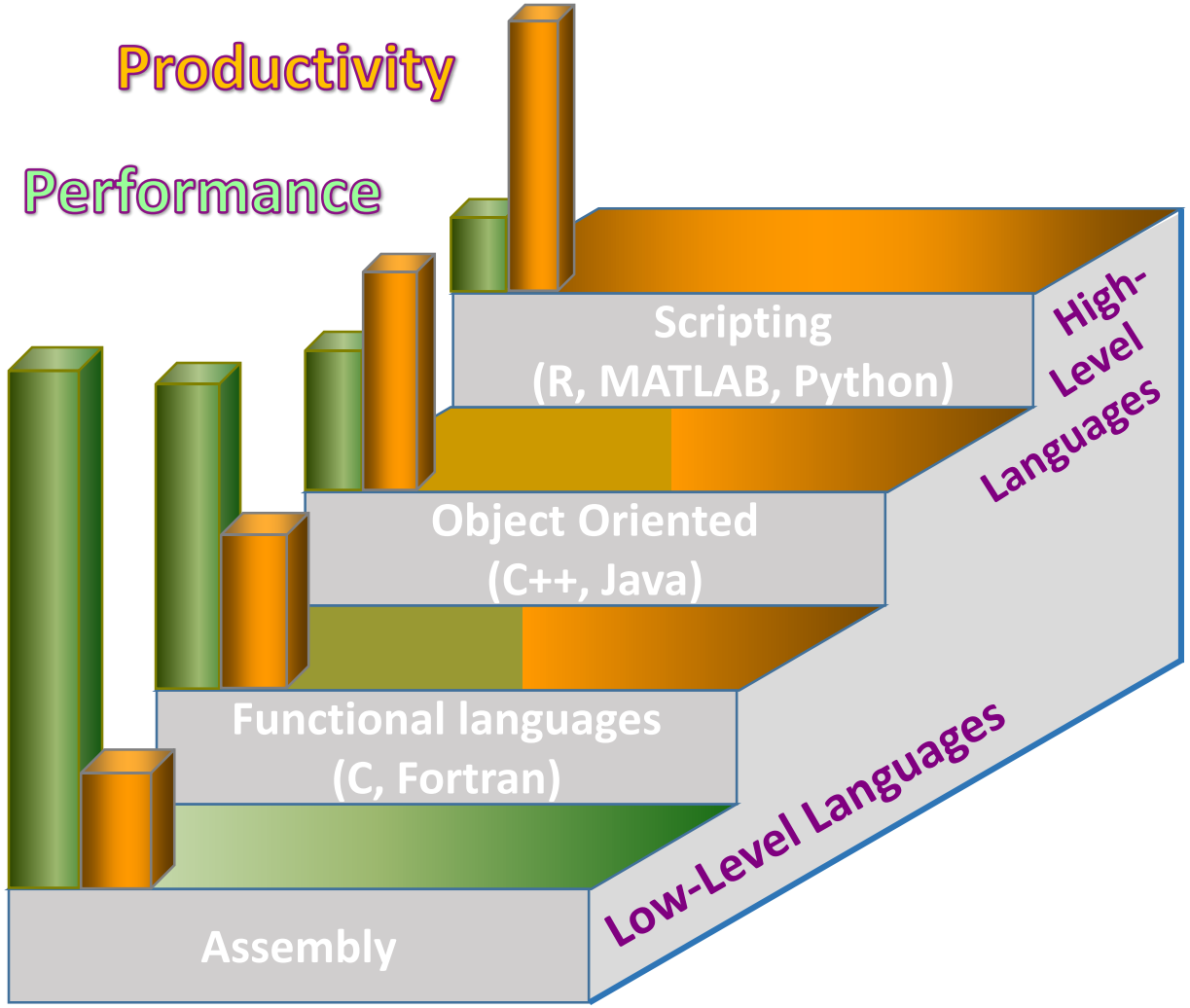
Introduction to R

What is R?

- R is a freely available language and environment for data manipulation, statistical computing and graphics.
- An interpreted computer language.
- Contributed packages expand the functionality to cutting edge research for more general data mining purpose.
- Most of the user-visible functions in R are written in R, calling upon a smaller set of internal primitives.
- Many statistical functions are already built in.
- Excellent graphical facilities for data analysis.

The Programmer's Dilemma



Increasingly large R package repository

- The Comprehensive R Archive Network (CRAN)
  - Currently, the CRAN package repository has ~8K packages
- This is an enormous advantage - new techniques available without delay, and they can be performed using the R language you already know.
- Allows you to build customized data mining and statistical programs suited to your own needs.
- Downside: as the number of packages grows, it is becoming difficult to choose the best package for your needs & QC is an issue.

Getting Started

- How to use help in R?
  - R has a very good help system built in.
  - If you know which function you want help with simply precede that function with ?.
    - E.g. ?hist
  - If you don’t know function name, then use help.search(“”) on the key word
    - E.g.: help.search(“histogram”).

R Objects

- Almost all things in R are OBJECTS.
  - Atomic variable types: numeric, character, boolean,
  - Vector
  - Array
  - List
  - Data frame
  - Function
  - Graphics are written out and are not stored as objects

Variables and Assignment

```
> num_var <- 49
> sqrt(num_var)
[1] 7
```

numeric

```
> char_var <- "The dog ate my homework"
> sub("dog","cat", char_var)
[1] "The cat ate my homework“
```

character string

```
> bool_var <- (1+1==3)
> bool_var
[1] FALSE
```

Logical

- Do not confuse = (assignment) with == (equality)
    - = is a command, == is a question

Vectors and vector operations

To create a vector: an ordered collection of data of the same type

```
# c() command to create vector x
 x=c(12,32,54,33,21,65)
# c() to add elements to vector x
 x=c(x,55,32)
```

```
# seq() command to create sequence of number
 years=seq(1990,2003)
# to contain in steps of .5
 a=seq(3,5,.5)
# can use : to step by 1
years=1990:2003;
```

```
 # rep() command to create data that follow a regular pattern
 b=rep(1,5)
```

To access vector elements:

```
# 2nd element of x
 x[2]
# first five elements of x
 x[1:5]
# all but the 3rd element of x
 x[-3]
 # values of x that are < 40
 x[x<40]
# values of y such that x is < 40
 y[x<40]
```

To perform operations:

```
 # mathematical operations on vectors
y=c(3,2,4,3,7,6,1,1)
 x+y; 2*y;  x*y;  x/y;  y^2
```

Matrices & matrix operations

To create a matrix: : a two-dimensional rectangular table of data of the same type

```
# matrix() command to create matrix A with rows and cols
A=matrix(c(54,49,49,41,26,43,49,50,58,71),nrow=5,ncol=2))
B=matrix(1,nrow=4,ncol=4)
```

To access matrix elements:

```
# matrix_name[row_no, col_no]
 A[2,1]     # 2nd row, 1st column element
 A[3,]   # 3rd row
 A[,2]      # 2nd column of the matrix
 A[2:4,c(3,1)]   # submatrix of 2nd-4th
elements of the 3rd and 1st columns
 A["KC",] # access row by name, "KC"
```

Statistical operations:

```
 rowSums(A)
 colSums(A)
 rowMeans(A)
 colMeans(A)
# max of each columns
 apply(A,2,max)
# min of each row
 apply(A,1,min)
```

Lists

- Objects containing an ordered collection of objects
- Components do not have to be of same type
- Use *list()* to create a list:
  - *a <- list(“hello”,c(4,2,1),“class”);*
- Components can be named:
  - *a <- list(string1=“hello”,num=c(4,2,1),string2=“class”)*
- Use [[position#]] or $name to access list elements
  - E.g., a[[2]] and a$num are equivalent
- Running the *length()* command on a list gives the number of higher-level objects

Data frames

- represent the typical record dataset that researchers usually encounter – like a spreadsheet.
- Rectangular table with rows and columns;
  - data within each column has the same type (e.g. number, text, logical), but different columns may have different types.
- Understand dataset
  - Summary(dataframe)
  - Str(dataframe)

Reading data from files

- Reading a table of data can be done using the read.csv() or *read.table()* command:
  - *a <- read.csv(“a.csv”)*
  - *b <- read.table(“b.txt”)*
- The values are read into R as an object of type data frame. Various options can specify reading or discarding of headers and other metadata.
- A more primitive but universal file-reading function exists, called *scan()*
  - b <- scan(“input.dat”);
  - *scan()* returns a vector of the data read

Data Subsetting

- Use a logical operator to do this.
  - ==, >, <, <=, >=, <> are all logical operators.
  - Note that the “equals” logical operator is two equal signs (==).
- Example:
  - `D[D$Gender == “M”,]`
  - This will return the rows of D where Gender is “M”.
  - Remember R is case sensitive!
  - This code does nothing to the original dataset.
  - `D.M <- D[D$Gender == “M”,]` gives a dataset with the appropriate rows.

Writing your own functions

- Writing functions in R is defined by an assignment like:
  - *Function_name <- function(arg1,arg2) { function_commands; }*
- Functions are R objects of type “function”
- Arguments may have default values
  - Example: *my.pow <- function(base, pow = 2) {return base^pow;}*
  - Arguments with default values become optional, should usually appear at end of argument list (though not required)
- Arguments are untyped
  - Allows multipurpose functions that depend on argument type
  - Use *class()*, *is.numeric()*, *is.matrix()*, etc. to determine type

Conditional statements

- Perform different commands in different situations
- *if (condition) command_if_true*
  - Can add *else command_if_false* to end
  - Group multiple commands together with braces {}
  - *if (cond1) {cmd1; cmd2;} else if (cond2) {cmd3; cmd4;}*
- Conditions use relational operators
  - ==, !=, <, >, <=, >=
- Combine conditions with *and* (&&) and *or* (||)

Loops

- Most common type of loop is the *for* loop
  - *for (x in v) { loop_commands; }*
  - *v* is a vector, commands repeat for each value in *v*
  - Variable *x* becomes each value in *v*, in order
  - **Example:** adding the numbers 1-10
    - *total = 0; for (x in 1:10) total = total + x;*
- Other type of loop is the *while* loop
  - *while (condition) { loop_commands; }*
  - Condition is identical to *if* statement
  - Commands are repeated until condition is false
    - Might execute commands 0 times if already false
- *while* loops are useful when you don't know number of iterations

Scripting in R

- A script is a sequence of R commands that perform some common task
  - E.g., defining a specific function, performing some analysis routine, etc.
- Save R commands in a plain text file
  - Usually have extension of .R
- Run scripts with *source()* :
  - *source("filename.R")*
- To save command output to a file, use *sink()*:
  - *sink("output.Rout")*
  - *sink()* restores output to console
  - Can be used in or outside of a script

R Packages

- R functions and datasets are organized into packages
  - Packages *base* and *stats* include many of the built-in functions in R
  - CRAN provides thousands of packages contributed by R users
- Packages must be installed before they can be loaded
  - Use *library()* to see installed packages
  - Use *install.packages("pkgname")* and *update.packages("pkgname")* to install or update a package
- Package contents are only available when loaded
  - Load a package with *library(pkgname)*

Useful R links

- R Home: http://www.r-project.org/
- R's CRAN package distribution: http://cran.cnr.berkeley.edu/
- Introduction to R manual: http://cran.cnr.berkeley.edu/doc/manuals/R-intro.pdf
- Writing R extensions: http://cran.cnr.berkeley.edu/doc/manuals/R-exts.pdf
- Other R documentation: http://cran.cnr.berkeley.edu/manuals.html