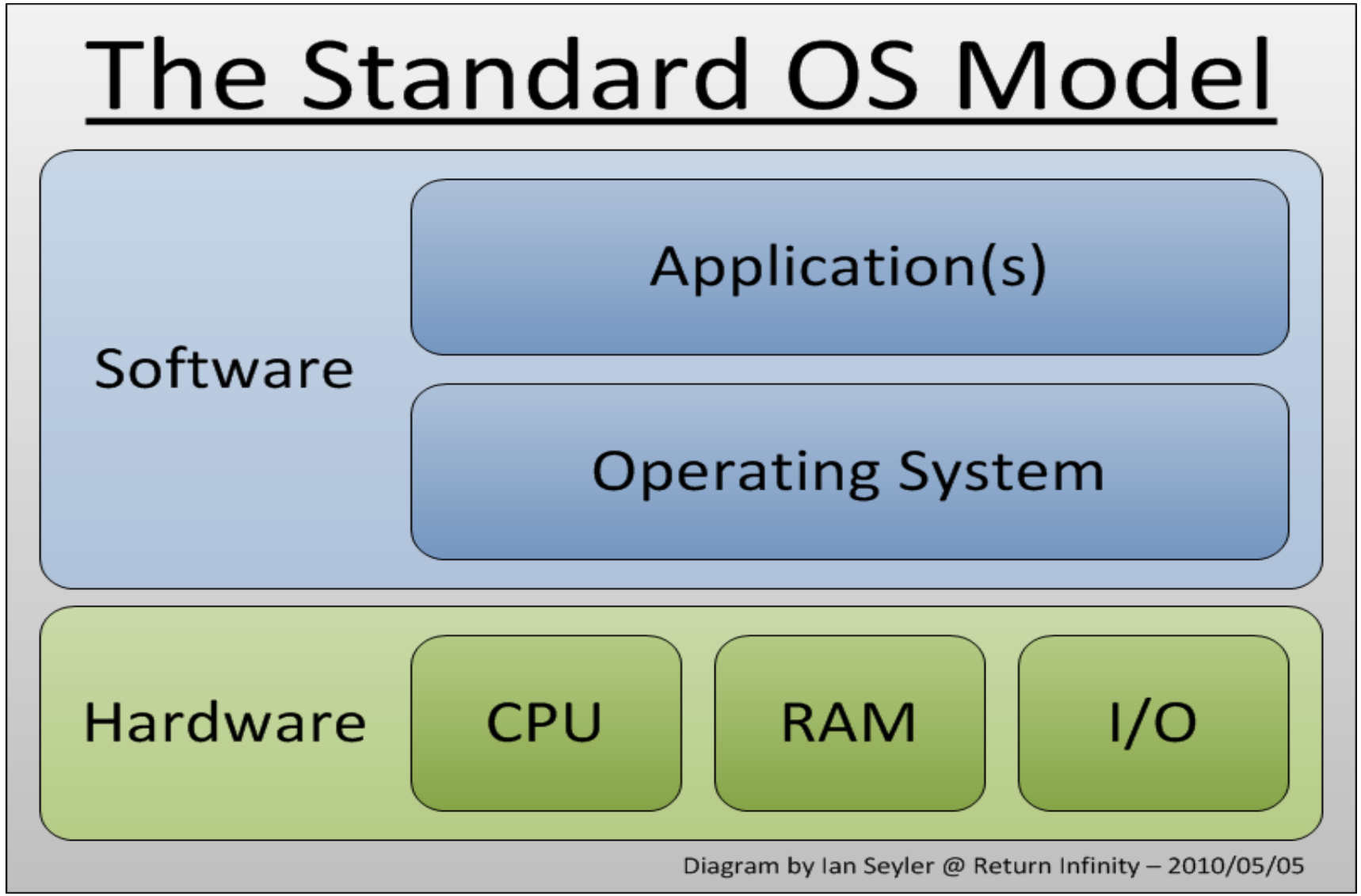
Intro to R Programming

-- Nikhil Vidhani

What this course is about?

- Basics of Computer Architecture and Programming
- Intro to Programming through R
- Popular R methods and their use in Data analysis
- Technical Documentation: Some tips for Word, Latex and R-markdown.

What's a computer look like?



What does it do?

- Perform Calculations!
  - Billions of them every second.
  - Cores, threads, clock speed
- Stores data
  - Cache vs RAM vs HDD
  - Speed vs storage cost
- Runs Software
  - System (OS): Linux, Windows and Mac-OS
  - Application: R, RStudio, Excel

What is a program?

- Translation of an algorithm into a language that computer understands
- An algorithm takes input, perform some operations and gives output
  - Executes in finite time
  - E.g. sorting, searching, reading, copying!

- Complexity of a Program
  - Time and space!
  - E.g. Fibonacci series!

- Programming Paradigms
  - Iterative vs Recursive
  - Procedural vs Object Oriented

- Good Program
  - Re-readable, organized and modular

Typical Programing Errors

- Syntactical (spelling mistake)
  - Will get caught very easily! Just run the program.

- Semantic Errors (meaningless operations)
  - For e.g. “nikhil”+32
  - Exceptions: like divide by 0.
  - May get caught. A warning will be thrown nonetheless.

- Logical Errors (Unintentional)
  - Program will crash, run forever or give a wrong answer!
  - Debugging requires some skill and experience.

What is R

- Implementation of S Programming language
  - Started as statistical environment
  - Explains the deep rootedness of R in statistics
  - Mostly written in C (earlier FORTRAN)
  - More info on Wikipedia!

- Philosophy behind R (or S, S+)
  - Interactive environment
  - Transition from users to Programmers as per need!
  - You don’t need to be a programmer to learn (and) use basic R
  - More info at http://ect.bell-labs.com/sl/S/history.html

What is R (cont.)

- Features
  - Very easy to follow and understand
    - Require understanding of vector and matrix indexing!
    - Interactive
  - Runs on all platforms.
    - Small software to download and load. Use packages as per need.
  - Free of cost. Open source software (GNU GPL). More info at www.fsf.org
  - Very active development
    - Frequent updates and releases
    - Very active and responsive user community – Stackoverflow!

- Drawbacks
  - Limited 3-D graphics capability
  - Everything must be in RAM – big data?
  - If a functionality is missing you got to code it yourself!

What if not R

- Closest cousin is MATLAB
  - Although used much more in engineering than in statistics
  - Syntax is similar to R (Read: http://mathesaurus.sourceforge.net/octave-r.html)
  - Python is also very popular although its more meaningful for data science

- Statistical Alternatives?
  - SAS and Stata
  - Both are paid software
  - Very different than R in syntax!
    - Non-interactive
    - Limited user community support
  - Despite the differences Stata is very popular in management research. And there are some die-hard SAS fans in Finance too.

Downloading and Installing R

- Download R: https://cran.r-project.org/
  - Choose base package for your OS
    - Windows: https://cran.r-project.org/bin/windows/base/R-3.5.0-win.exe
    - Linux: Use apt-get (Debian based) OR yum install (RPM based) from terminal.
    - Mac: https://cran.r-project.org/bin/macosx/R-3.5.0.pkg
  - Install R

- Download RStudio IDE
  - Choose the free RStudio Desktop edition
  - https://www.rstudio.com/products/rstudio/download/#download
  - Choose the appropriate one according to your OS
  - Install RStudio

Getting Help in R

- From Console
  - Just type: ? followed by function name without parenthesis
  - E.g. `?mean; ?sum; ?length;`
  - Clarify:
    - **?mean** - help for the function “mean”
    - **??mean** - will perform the search over the internet (CRAN database)
      - Look for base::mean!
    - **mean()** - call the function mean
    - **mean** - print the definition of the function “mean”

- From Web sources
  - Most reliable and easy to incorporate is www.stackoverflow.com.
  - www.r-bloggers.com is also quite helpful.
  - You can use https://cran.r-project.org for any resource on R
  - Even typing your question in google will get you good results!
    - 99% of your questions are already answered! You just need to find them!

R Input and Output

- Simple assignment
  - `X = 1;` (*or* `X <- 1;`)
  - Assignment is always right to left
    - Read 1 goes into X
    - We aren't comparing X with 1 here
  - The semi-colon isn't necessary in R, but it's a good practice to use it
  - `X = ;` is incomplete
  - # (prefix) is used as a comment. Use it for helpful comments.
  - Use Ctrl-Shift-C for multi-line comments

- Value of X can be seen by
  - `X;`

Vectors

- A sequence of numbers. Many ways to input!
  - `Y = c(1,7,-3,41); # concatenate arbitrary numbers`
  - `Y = 1:10; # natural numbers`
  - `Y = seq(1,100,9); # skip by 9`
  - `Y = rep(2, 3); # repeat 3 times`
  - `Y = rep(1:2, 3); # repeat the vector`
  - `Y = rep(1:2, each = 3); # repeat each element 3 times`
  - `Y = c(); # empty vector`
  - Execute this: `c(1:3, rep(c(5,7), each = 2), rep(9, 4), 7);`

- Length of vector: `length(Y);`

- Accessing i^{th} element of vector: `Y[i]; # square brackets`
  - `i` should be between `1` and `length(Y)`
  - Printing the entire vector is as before: `Y;`

Objects in R

- 5 basic (atomic) types of objects
  - character – strings
  - numeric – real numbers. Also called double.
  - integer – natural numbers. Default data type for numeric vectors.
    - `typeof(1:10)`
  - complex – complex numbers. We won't use them now!
  - logical – True/False (binary)

- Most basic collection of objects is a vector (also called an array)
  - Can only contain objects of same class (i.e. character or integer; not both)
  - "list" is a special type of object and can contain heterogeneous objects
    - Any Combination of vector, matrix, atomic types etc.
    - It can even contain another list as an object. E.g. linked-lists!
    - Due to its generality its very slow and hence rarely used with large datasets unless situation demands it

Numbers

- Default type of any number is numeric (i.e. real). `typeof(1)`
- R can differentiate between corner cases:
  - `1/0` is `Inf` -- `is.infinite();`
  - `0/0` is `NaN` -- `is.nan();`
  - Missing data is `NA` -- `is.na();`
  - Check what's `Inf-Inf` ?

- Arithmetic Operations
  - `*` multiplies
  - `/` divides
  - `^` takes exponent
  - `%%` is the modulo (remainder) operator. Try: `7 %% 2;`

Coercion

- Mixing Objects
  - Automatically coerced to the same class.
  - Try: `c(1:7, "a"); c(T, 2); c("a", FALSE);`
  - Implicit coercion!
  - Never use unless you know what you're doing!

- Explicit Coercion
  - `as.character(1:5);`
  - `as.numeric("iimb"); # warning!`
  - `as.logical(seq(-2,2,1));`

List

- Can carry different types of data together
  - `L = list(1, FALSE, 3.14, "iimb", "c", 4-3i);`
  - Print list: `L;`
  - L is in fact a list of lists. Check: `typeof(L); typeof(L[4]); typeof(L[[4]]);`
  - Single square brackets `[i]` access the i^{th} list embedded in the list `L`
  - Double square brackets `[[i]]` access the i^{th} element
  - Can append elements in list: `L = append(L, "7th");`
  - `unlist(L);` will coerce all elements into a single type and return a vector
  - Delete an element from a list:
    - I don't know how to do that!
    - Let's google: "delete element from list in R"
    - Open the answer on www.stackoverflow.com

Matrices

- Generalization of vectors
  - 2 dimensions instead on one!
  - N x K matrix means a matrix having N rows and K columns. Total of NK elements.
  - `M = matrix(nrow = 2, ncol = 3);`
  - Dimensions: `dim(M);`
  - Can think of M as
    - 3 columns vectors each of length 2, *or*
    - 2 row vectors each of length 3
  - Populate matrix: `M = rbind(1:3, 4:6);`
  - Alternatively populate as: `M = cbind(1:2, 3:4, 5:6);`

Matrices (cont.)

- Indexing a matrix
  - `M[i,j]` gives the element at i^{th} row and j^{th} column
  - `M[i,]` gives the entire i^{th} row (a vector)
  - `M[,j]` gives the entire j^{th} column (a vector)

- Matrix multiplication
  - `*` just does an element wise multiplication, i.e. $(M * M)_{ij} = M_{ij} * M_{ij}$
  - `%*%` performs the usual matrix multiplication. Try: `M %*% M`
    - Dimensions must match
    - Try `t(M) %*% M;`
    - `t(M)` takes transpose of a matrix!

Matrices (cont.)

- Identity matrix: `diag(3)`
- Diagonal Matrix: `diag(c(1,5,7)); diag(1:7);`
- Diagonal of a matrix: `diag(M)`
- Trace of a matrix: `sum(diag(M))`
- Inverse of a matrix:
  - Must be a square matrix: `M = matrix(1:9, nrow = 3, ncol = 3);`
    - Another way to create a matrix. Data is entered column-wise.
  - Determinant must be non-zero: `det(M); M[3,3] = 19; det(M);`
  - Inverse: `solve(M);`

Factors

- For categorical data.
  - Male, female
  - Cities in a dataset
  - Typically useful when the dataset is large but the no. of categories is small
  - Very useful in the regression framework using `lm();`
    - Automatically creates dummy for all but one categories.
  - Using factors is more descriptive than integer values
    - Rather than using 1 for PGP, 2 for FPM and 3 for Others; its more intuitive to use factors.
  - Example:
    - `sex = rep(c("male", "female"), 5);`
    - `sex_f = as.factor(sex);`
    - `Check: typeof(sex_f); as.integer(sex_f);`

Data Frame

- Probably the most important data type you'll use.
  - All external data (from excel, csv, tables, webpages etc) is read as data frame
  - It's a list where each element of list must have the same length.
  - Think of it like a matrix but with the flexibility that each column can have different data type. E.g. set of Names, weights and heights
  - Example:
    - `d = data.frame(name = c("a", "b"), weight = c(70, 75), height = c(1.78, 1.82));`
    - `d; d$name; d[1,]; d$weight; d[,3];`
    - `d$bmi = d$weight / (d$height)^2;`
    - `nrow(d); ncol(d); dim(d);`
    - `colnames(d)[1] = "names";`
    - `rownames(d) = c("I", "II");`

Reading Data

- Download some stock data from NSE
  - https://www.nseindia.com/products/content/equities/indices/historical_index_data.htm
  - Save the CSV file as data.csv
- From CSV (most common)
  - `setwd("D:/Opera Downloads/"); nifty = read.csv("data.csv");`
  - Alternatively: `nifty = read.csv("D:/Opera Downloads/data.csv");`
- From Excel
  - Search it yourself! It is not recommended btw.
- From clipboard
  - `read.table("clipboard");`
  - This is quick fix for small data transfer between R and excel. Use `read.csv()` as your primary method for data reading!

Reading Data (cont.)

- Viewing data
  - `View(nifty);`
- Date
  - `nifty$Date = as.Date(nifty$Date, format = "%d-%b-%Y");`
  - `n = nrow(nifty);`
  - `d = nifty$Date[1];`
  - `format(d, format = "%D");  # 04/02/18`
  - `format(d, format = "%d-%m-%y");  # 02-04-18`
  - `format(d, format = "%d.%b.%Y");  # 02.Apr.2018`
  - `format(d, format = "%d_%B_%Y");  # 02_April_2018`
- Alternatively,
  - `read.table("data.csv", header = T, sep = ",", nrows = 5);`

if-else

-
```
if(<COND_1>) {
  # do something!
}
```

-
```
if(<COND_1>) {
  # do something!
} else {
  # ...
}
```

-
```
if(<COND_1>) {
  # do something!
} else if(<COND_2>) {
  # ...
} else {
  # ...
}
```

```
if(nifty$Close[2] > nifty$Close[1]) {
  str = paste("Stock market closed green on", nifty$Date[2]);
} else if(nifty$Close[2] > nifty$Open[2]) {
  str = paste("Stock market closed above opening on", nifty$Date[2]);
} else {
  str = paste("Stock market was red and closed below opening on", nifty$Date[2]);
}
print(str);
```

for loop

- Looping is used to perform similar set of tasks repetitively
  - ```
    for(i in n:1) {
      print(nifty$Date[i]);
    }
    ```
  - `n:1;` is same as `seq(n,1,1);` i.e. backwards counting!
  - Alternatively, you can execute: `rev(nifty$Date);` or `nifty$Date[n:1];`
- Try avoiding loops if you can!
  - Increasing all dates by a week: `nifty$Date + 7`
  - Finding Daily growth: `nifty$Close[-1] / nifty$Close[-n]`
  - Daily diff. b/w high and low prices: `nifty$High - nifty$Low`
  - Question: find % growth in daily volatility
    - Volatility is defined as: $Vol_t = (High_t - Low_t)/Open_t$
    - Percentage Growth is defined as: $\%G = \frac{(Value_{t+1} - Value_t)}{Value_t} * 100$
    ```

Nested if-else and for loop

```
for(i in 2:n) {
  if(nifty$Close[i] > 1.01 * nifty$Close[i-1]) {
    # market gained more than 1%
    for(j in 1:ncol(nifty)) {
      print( paste("Gain", i, colnames(nifty)[j], nifty[i,j], sep =":") );
    } # end for(j)
  } else if(nifty$Close[i] < 0.99 * nifty$Close[i-1]) {
    # market lost more than 1%
    for(j in 1:ncol(nifty)) {
      print( paste("Loss", i, colnames(nifty)[j], nifty[i,j], sep =":") );
    }
  } else {
    print(paste("Market movement was within 1% for i =", i));
  } # end if()
} # end for(i)
```

Jumping

- Till now all our commands executed sequentially
- There may be circumstances when we need to jump
- Next and Break
  - `next` is used to skip an iteration, while `break` exits the loop entirely.
  - ```
    for(i in 1:10) {
      if(i <= 3) {
        next;
      }
      if(i > 6) {
        break;
      }
      print(i);
    }
    i;
    ```
- `return()` is used to exit a function with a value.
    ```