Advanced R Programming - Lecture 5

Krzysztof Bartoszek, Shashi Nagarajan
(slides based on Leif Jonsson's and Måns Magnusson's)

Linköping University

krzysztof.bartoszek@liu.se

16 IX 2022 (S26)

Today

Input and output

Basic I/O

Cloud storage

web APIs: Lab

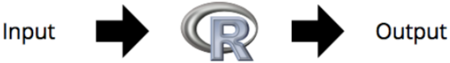
web scraping

Shiny

Relational Databases

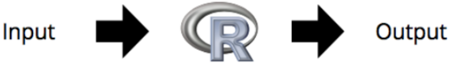
Questions since last time?

Input and output



Format, localization

Input and output



Format, localization and encoding...... hell!

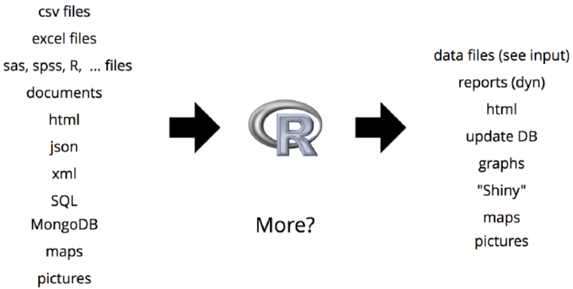
`http://www.joelonsoftware.com/articles/Unicode.html`
The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)

Unicode defines codes for **all (?)** characters—multiple encodings (for a given language only small fraction of characters used)

`Content-Type` tag for HTML

BUT e–mail, `.txt`, `.csv`

" Formats"



csv files

excel files

sas, spss, R, ... files

documents

html

json

xml

SQL

MongoDB

maps

pictures

More?

data files (see input)

reports (dyn)

html

update DB

graphs

"Shiny"

maps

pictures

Localization

🏠	🌐
own Computer	Cloud Storage
local network	web pages
local database	web scraping
	web APIs
	remote database

Table: Local - Remote

Files on your computer

```
# Input simple data
read.table()
read.csv()
read.csv2()

load()


# Output simple data
write.table()
write.csv()
write.csv2()

save()
```

More complex formats

software/data	**package**
Excel	XLConnect
SAS, SPSS, STATA, ...	foreign
XML	xml
JSON (GeoJSON)	rjsonio, RJSON
Documents	tm
Maps	sp
Images	raster

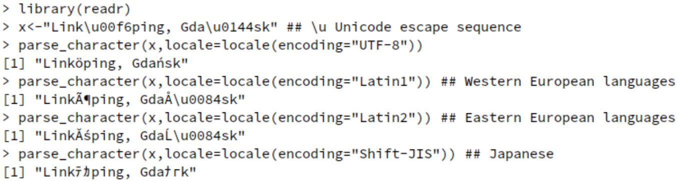
Table: Format - R package

Format issues examples

Data stored as column names

```
>cbind("id"=c("A","B"),"2020"=c(1,2),"2021"=c(5,6))
     name 2020 2021
[1,] "A"  "1"  "5"
[2,] "B"  "2"  "6"
```

Different encodings



```
> library(readr)
> x<-"Link\u00f6ping, Gda\u0144sk" ## \u Unicode escape sequence
> parse_character(x,locale=locale(encoding="UTF-8"))
[1] "Linköping, Gdańsk"
> parse_character(x,locale=locale(encoding="Latin1")) ## Western European languages
[1] "LinkÃ¶ping, GdaÅ\u0084sk"
> parse_character(x,locale=locale(encoding="Latin2")) ## Eastern European languages
[1] "LinkĂśping, GdaĹ\u0084sk"
> parse_character(x,locale=locale(encoding="Shift-JIS")) ## Japanese
[1] "Linkﾃｶping, Gdaﾅгk"
```

Cloud storage

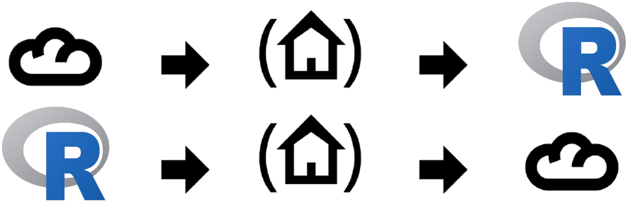


Table: Local - Remote

Why?

Robust

Backups

Cloud computing

can be tricky in the beginning

but

Why?

Robust

Backups

Cloud computing

can be tricky in the beginning

but how about safety? (data leaks, outsourcing)

But control on what is going on? (outsourcing, denial of service)

BUT

Why?

Robust

Backups

Cloud computing

can be tricky in the beginning

but how about safety? (data leaks, outsourcing)

But control on what is going on? (outsourcing, denial of service)

BUT requires internet connection

Localization

Arbitrary data

Structured data

API Packages

Remote	package
General	downloader
GitHub	repmis, downloader
Dropbox	rdrop
Amazon	RAmazonS3
Google Docs	googlesheets

web APIs

application program interface using http

"contract to 'get data' online"

more and more common

examples:

github

Riksdagen

Statistics Sweden

RESTful

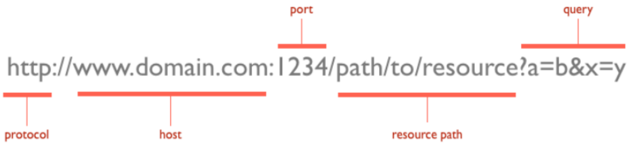
Basic principles:

Data is returned (JSON / XML)

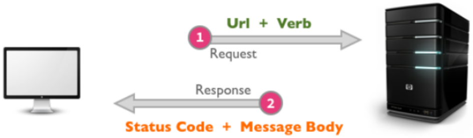
Each specific data has its own URI

Communication is based on HTTP verbs

Hypertext Transfer Protocol (http)



Hypertext Transfer Protocol (http)



Verbs

Verb	Description
GET	Get "data" from server.
POST	Post "data" to server (to get something)
PUT	Update "data" on server
DELETE	Delete resource on server

Status codes

Code	Description
1XX	Information from server
2XX	Yay! Gimme' data!
3XX	Redirections
4XX	You failed
5XX	Server failed

Example REST API's

`https://www.naturvardsverket.se/amnesomraden/luft/statistik--utslapp-och-halter/luftkvaliteten-i-realtid-och-preliminar-statistik/webbtjanster-luftkvalitetsdata`
Air quality in Sweden API

`https://developers.google.com/maps/documentation/geocoding/intro`
Google Map Geocode API

Common API formats

JavaScript Object Notation (JSON)
Think of named lists in R
R Packages: RJSONIO, rjsonlite

Extensible Markup Language (XML)
Older format (using nodes)
xpath
R Packages: XML

JSON

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
        "streetAddress": "21␣2nd␣Street",
        "city": "New␣York",
        "state": "NY",
        "postalCode": "10021"
  },
  "phoneNumber": [
        { "type": "home", "number": "212␣555" },
        { "type": "fax", "number": "646␣555" }
  ],
  "newSubscription": false,
  "companyName": null
}
```

XML

```
<?xml version="1.0" encoding="utf-8"?>
<wikimedia>
<projects>
<project name="Wikipedia" launch="2001-01-05">
<editions>
<edition language="English">en.wikipedia.org</edition>
<edition language="German">de.wikipedia.org</edition>
<edition language="French">fr.wikipedia.org</edition>
<edition language="Polish">pl.wikipedia.org</edition>
<edition language="Spanish">es.wikipedia.org</edition>
</editions>
</project>
<project name="Wiktionary" launch="2002-12-12">
<editions>
<edition language="English">en.wiktionary.org</edition>
<edition language="French">fr.wiktionary.org</edition>
<edition language="Vietnamese">vi.wiktionary.org</edition>
<edition language="Turkish">tr.wiktionary.org</edition>
<edition language="Spanish">es.wiktionary.org</edition>
</editions>
</project>
</projects>
</wikimedia>
```

web scraping

Unstructured http(s) data

Often HTML format

Spiders / scraping / web crawlers

Basics behind search engines

HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>This is a title</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

(har)rvest

JavaScript Object Notation (JSON)
Simplify spider activity

Download data
Parse data
Follow links
Fill out forms
Store crawling history

Difficulties and bad spiders

Scraping is fragile!
Difficulties and bad spiders
www.domain.se/robot.txt
Politeness

robot traps
javascript
delays

Shiny

- Shiny is an R package that makes it easy to build **web apps** (both local and internet based ones) using R
- A major use-case: Interactive **dashboards** for data visualisation/analysis in commercial (& research) applications
- Shiny allows users **without** advanced web-development skills to build apps with User Interfaces (UIs) that are *reactive* to user actions; in essence, Shiny generates HTML code from R
- Advanced web-developers can, however, extend Shiny apps to have CSS themes, htmlwidgets and JavaScript actions; to add your own HTML to the UI, you could use the `HTML()` function

See `https://www.rstudio.com/products/shiny/shiny-user-showcase/` for a number of Shiny app examples

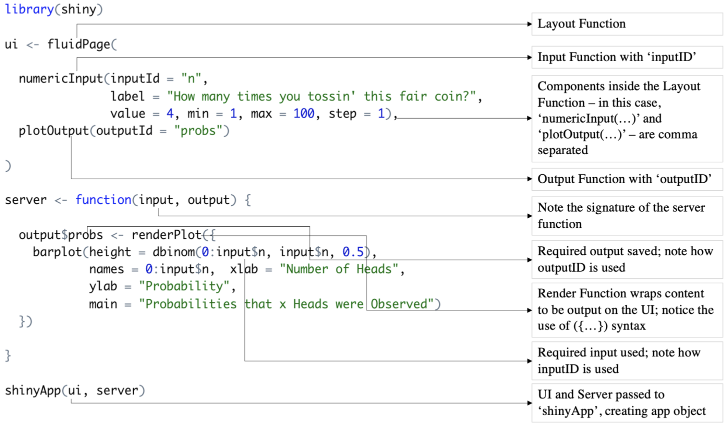
Building a Basic Shiny App: I

- Each Shiny App must have two components – a **UI** *object* (the app's 'frontend') and a **server** *function* (its 'backend')
- The UI is written out as a **layout function**, which is typically customised with one or more **input** and **output** functions
- The server function must perform all needed data-access operations (if any), possibly using the **inputs** obtained through the UI and assign as **output** everything to be displayed on the UI, after wrapping them inside constructs such as **render** functions
- The server function can take up to three parameters: `input`, `output`, and another variable to manage multiple user sessions, called `session`

Building a Basic Shiny App: II

- The UI and server components can both be written in a *single* .R file or *two separate* .R files, depending upon your preference
- The Shiny app object, as such, is created when the UI and server components are passed to a function such as `shiny::shinyApp` or `shiny::runGitHub`
- **Important**: When writing a Shiny app as an R package (possibly for publication and distribution), either store the .R file(s) with the UI/server components in the root package directory (or) in the package's `inst` folder
- **Tip**: When trying to subset data, especially data containing missing values, consider using the `subset` or `dplyr::filter` functions!

A Simple Example worth Studying: I



```r
library(shiny)

ui <- fluidPage(

  numericInput(inputId = "n",
             label = "How many times you tossin' this fair coin?",
             value = 4, min = 1, max = 100, step = 1),
  plotOutput(outputId = "probs")

)

server <- function(input, output) {

  output$probs <- renderPlot({
    barplot(height = dbinom(0:input$n, input$n, 0.5),
        names = 0:input$n,  xlab = "Number of Heads",
        ylab = "Probability",
        main = "Probabilities that x Heads were Observed")
  })

}

shinyApp(ui, server)
```

Layout Function

Input Function with ‘inputID’

Components inside the Layout Function – in this case, ‘numericInput(…)’ and ‘plotOutput(…)’ – are comma separated

Output Function with ‘outputID’

Note the signature of the server function

Required output saved; note how outputID is used

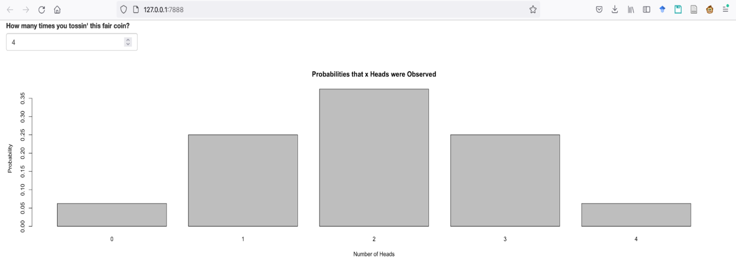
Render Function wraps content to be output on the UI; notice the use of ({…}) syntax

Required input used; note how inputID is used

UI and Server passed to ‘shinyApp’, creating app object

Try playing around with the app; code shared here ./shiny_app.R

A Simple Example worth Studying: II



127.0.0.1:7888

How many times you tossin' this fair coin?

4

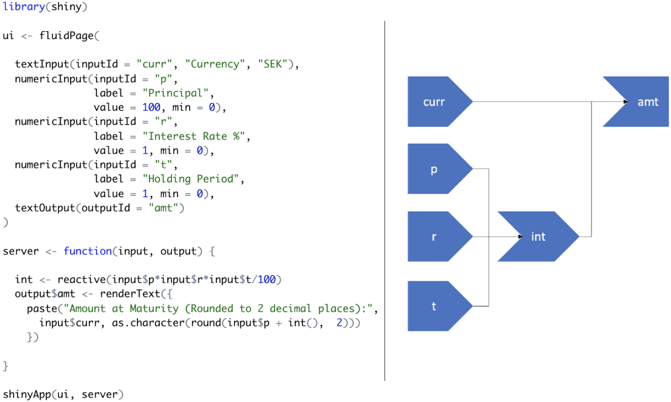
Figure: Screenshot of the Example App

- `numericInput` within the `ui` function is an example of **input validation**; app users can only input integers between 1 & 100
- App objects generated by `shinyApp(ui, server)` open up on a web **browser** by default

Reactivity (reactive programming): I

- Shiny allows us to build **reactive** apps, where outputs can change whenever a user manipulates the UI to change an input
- For reactivity, however, the *server* function must always access UI inputs within a **reactive context** created by constructs such as **render** functions or the **reactive** function
- Similarly, content assigned to UI output(s) for display **must** be wrapped inside reactive context(s)
- Do review the example from earlier to see these rules obeyed!
- Internally, Shiny manages reactivity by creating a **reactive graph** capturing relationships (called **reactive dependencies**) between input and output objects and executing reactive interactions in a **lazy** manner

Reactivity (reactive programming): II



```
library(shiny)

ui <- fluidPage(

  textInput(inputId = "curr", "Currency", "SEK"),
  numericInput(inputId = "p",
               label = "Principal",
               value = 100, min = 0),
  numericInput(inputId = "r",
               label = "Interest Rate %",
               value = 1, min = 0),
  numericInput(inputId = "t",
               label = "Holding Period",
               value = 1, min = 0),
  textOutput(outputId = "amt")
)

server <- function(input, output) {

  int <- reactive(input$p*input$r*input$t/100)
  output$amt <- renderText({
    paste("Amount at Maturity (Rounded to 2 decimal places):",
      input$curr, as.character(round(input$p + int(),  2)))
    })

}

shinyApp(ui, server)
```

Figure: **Left**: Code for another reactive Shiny app - it computes the amount at maturity of a principal locked in at a fixed simple interest rate (see `./reactive_app.R`); **Right**: the Reactive Graph corresponding to this app - `curr`, `p`, `r` and `t` are the input variables Shiny tracks ('listens') to; `int` is an intermediate output variable produced by Shiny and `amt` is the output variable which connects back to the `ui`; black arrows connect these variables according to the `server` function; note the shapes of the input, intermediate output and output variables in the graph

Publishing your Shiny App

locally
zip-file in cloud
github (see runGithub())

Publishing your Shiny App

🏠

locally
zip-file in cloud
github (see runGithub())

🌐

your own server
shinyapps.io

Relational Databases

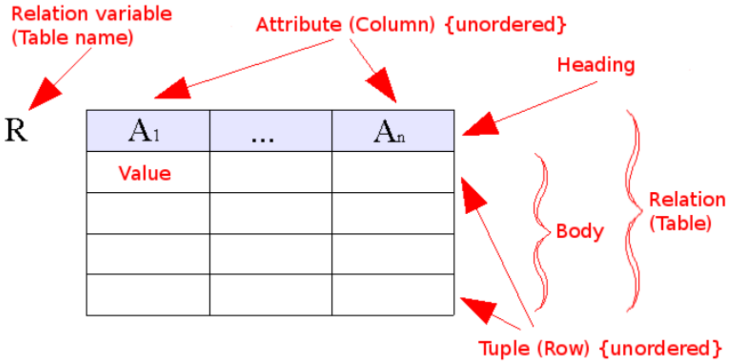
Structured database in tables

local or online

query language for I/O

effective for big data

difficult to design


Relation variable
(Table name)
Attribute (Column) {unordered}
Heading
R
A1
...
An
Value
Body
Relation
(Table)
Tuple (Row) {unordered}

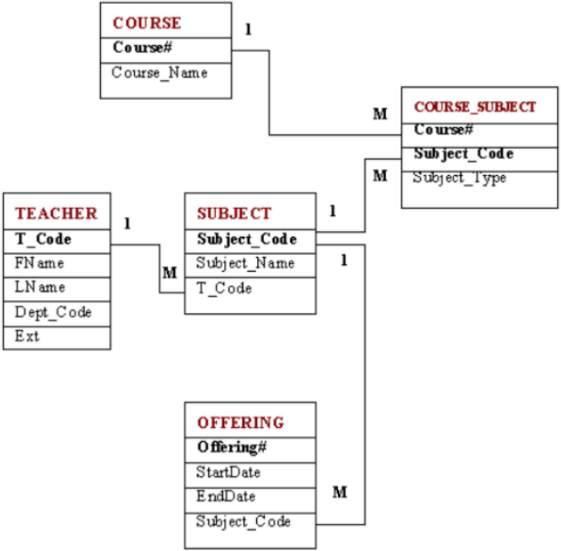
Keys

Superkey "set of attributes such that two distinct rows do not have the same values for these attributes"

Primary key (attribute): choice of superkey, relationships between tables are done through the primary key

`https://en.wikipedia.org/wiki/Superkey`

`https://en.wikipedia.org/wiki/Primary_key`


COURSE
Course#
Course_Name
1
M
COURSE_SUBJECT
Course#
Subject_Code
Subject_Type
M
TEACHER
T_Code
FName
LName
Dept_Code
Ext
1
M
SUBJECT
Subject_Code
Subject_Name
T_Code
1
1
OFFERING
Offering#
StartDate
EndDate
Subject_Code
M

A good database

Can be difficult to design ?

A good database

Can be difficult to design ?
No duplicates
No redundancies
Easy to update
"Normal forms"

A good database

Can be difficult to design ?
No duplicates
No redundancies
Easy to update
"Normal forms"

Easy to query

A good database: normalization

Database normalization: “is the process of restructuring a relational database in accordance with a series of so–called normal forms in order to reduce data redundancy and improve data integrity”
(usually divide table into separate tables linked by primary keys)

Denormalization: create redundancies for increased performance:
(preferred) store normalized data and allow DBMS to create additional redundancies (DBMS is responsible for inconsistencies)
(common) designed denormalized DB (designer is responsible for inconsistencies)

`https://en.wikipedia.org/wiki/Database_normalization`

`https://en.wikipedia.org/wiki/Denormalization`

A good database: normal forms in brief

First normal form: in each attribute (column) entry there is a single atomic value:
for Telephone Number you cannot have two telephone numbers

A good database: normal forms in brief

First normal form: in each attribute (column) entry there is a single atomic value:
for Telephone Number you cannot have two telephone numbers

Second normal form: 1NF and each non–primary attribute depends functionally only on the primary attribute and not on any other attribute:
(Course_code, Course_name, University, University_country) is not in 2NF as University_country is defined through University
here (Course_code, University) is the (composite) primary key

`https://en.wikipedia.org/wiki/X_normal_form`, X appropriate form

A good database: normal forms in brief

Third normal form: 2NF and “Every non–prime attribute of R is non–transitively dependent on every key of R.”: (University, Year, Vice–Chancellor, Vice–Chancellor DOB)
composite primary key (University,Year)
Vice–Chancellor DOB depends on key via Vice–Chancellor
(what if someone made a typo when entering a second time?)

A good database: normal forms in brief

Third normal form: 2NF and "Every non–prime attribute of R is non–transitively dependent on every key of R.": (University, Year, Vice–Chancellor, Vice–Chancellor DOB)
composite primary key (University,Year)
Vice–Chancellor DOB depends on key via Vice–Chancellor
(what if someone made a typo when entering a second time?)

Boyce–Codd normal form or 3.5NF: more strict than 3NF, no functional dependencies between two attributes of which neither is a superkey:
(city, land_plot, postal_code) fails due to relationship between city and postal_code

`https://en.wikipedia.org/wiki/X_normal_form`, X appropriate form

A good database: normal forms in brief

Fourth normal form: 3NF and no multiple multivalued dependencies:

(Teacher, Language, Course), primary key is whole entry

Version 1 (redundant)

KB, Polish, 732A94

KB, Polish, 732A63

KB, English, 732A94

KB, English, 732A63

KB, Swedish, 732A94

KB, Swedish, 732A63

Version 2 (what if I stop teaching R?)

KB, Polish, 732A94

KB, English, 732A94

KB, Swedish, 732A63

https://en.wikipedia.org/wiki/X_normal_form, X appropriate form

A good database: normal forms in brief

Fifth normal form: when there are complex constraints on the possible combinations of values

Sixth normal form: when there are temporal dependencies in data (can lead to table explosion)

A good database: normal forms in brief

Fifth normal form: when there are complex constraints on the possible combinations of values
Sixth normal form: when there are temporal dependencies in data (can lead to table explosion)

Domain–key normal form: values only constrained by permissable values for attributes and key uniquely identifying row: (Lecturer, Lecturer_description, University) fails (but 1NF?):
KB, **LiU** Statistician, **LiU**
TB, **SU** Mathematician, **SU**
TE, **LiU** Mathematician, **LiU**
FR, **SU** Biologist, **SU**

`https://en.wikipedia.org/wiki/X_normal_form`, X appropriate form

Using databases from R

Database system	**R package**
ODBC (Microsoft Access)	RODBC
PostgreSQL	RPostgresql
Oracle	ROracle
MySQL	RMySql
MongoDB	rmongodb

Table: Database - R package

The End... for today.

Questions?

See you next time!