The Mysterious Ellipsis: Tutorial

If you have any basic experience with R, you probably noticed that R uses three dots ellipsis (…) to allow functions to take arguments that weren't pre-defined or hard-coded when the function was built. Even though R beginners are usually aware of this behavior, especially due to some common functions that implement it (for example, paste()), they are often not using it enough in their own functions. In other cases, the ellipsis is just not used properly or not fully taken advantage of. In this tutorial we will go through some common mistakes in using the ellipsis feature, and some interesting options to fully utilize it and the flexibility that it offers.

Choose lists over vectors

The most common mistake is trying to assign the ellipsis content to a vector rather than a list. Well, of course it's not so much of a mistake if we're expecting only a single data type from the ellipsis arguments, but this is often not the case and assigning the arguments to a vector rather than a list might cause problems when there's a variety of data types.

So make sure you're always unpacking the ellipsis content using the list() function rather than the c() function. As an example, try running this piece of code with both options:

```
my_ellipsis_function <- function(...) {
args <- list(...) # good
# args <- c(...) # bad
length(args)
}
```

```
my_ellipsis_function(“Hello World”, mtcars)
```

Combine the ellipsis with other arguments

Some tend to think that it’s not possible to use the ellipsis with other regular arguments. This is not the case, and the ellipsis-arguments shouldn’t be the only ones in your function. You can combine them with as many regular arguments as you wish.

```
my_ellipsis_function <- function(x, ...) {
print(paste("Class of regular-argument:", class(x)))
print(paste("Number of ellipsis-arguments:", length(list(...))))
}

my_ellipsis_function(x = “Hello World”, mtcars, c(1:10), list(“Abc”, 123))
```

Don’t forget the names

In fact, the values of the arguments themselves are not the only information that is passed through the ellipsis-arguments. The names of the arguments (if specified) can also be used. For example:

```
my_ellipsis_function <- function(...) {
names(list(...))
}

my_ellipsis_function(some_number = 123, some_string = “abc”, some_missing_value = NA)
```

Lastly, somewhat of an advanced procedure might be unpacking the ellipsis-arguments into local function variables (or even global). There are all kind of scenarios where it might be needed (for global variables assignment it might be more intuitive). One example for a need in local variables, is where a certain function takes a certain regular-argument,

that is dependent on a varying set of other variables. A use of the function glue::glue() within another function is a good example for that. The following code demonstrates how simple it is to perform this “unpacking”:

```
my_ellipsis_function <- function(...) {
args <- list(...)

for(i in 1:length(args)) {
assign(x = names(args)[i], value = args[[i]])
}

ls() # show the available variables

# some other code and operations
# that use the ellipsis-arguments as “native” variables…
}

my_ellipsis_function(some_number = 123, some_string = “abc”)
```

So whether you’re an R beginners or not, don’t forget to utilize this convenient feature when needed, and use it wisely.

Modularize your Shiny Apps: Solutions

[emaillocker]Below are the solutions to these exercises on Shiny modules.

```
# load package
library(shiny)
```

```
####################
#                  #
#    Exercise 1    #
#                  #
####################
q1_module_UI <- function(id) {
  ns <- NS(id)
  selectInput(
    inputId = ns("letter"),
    label = "Select a letter",
    choices = LETTERS
  )
}

####################
#                  #
#    Exercise 2    #
#                  #
####################
q1_module <- function(input, output, session) {
  observe(print(input$letter))
}

####################
#                  #
#    Exercise 3    #
#                  #
####################
ui <- fluidPage(q1_module_UI(id = "q1"))

####################
#                  #
#    Exercise 4    #
#                  #
####################
server <- function(input, output, session) {
  callModule(module = q1_module, id = "q1")
}

shinyApp(ui = ui, server = server)
```

```
####################
#                  #
#    Exercise 5    #
#                  #
####################
q5_module_UI <- function(id) {
  ns <- NS(id)
  tagList(
    selectInput(
      inputId = ns("letter"),
      label = "Select a letter",
      choices = LETTERS
    ),
    textOutput(outputId = ns("selected_letter"))
  )
}

q5_module <- function(input, output, session) {
   output$selected_letter <- renderText(paste("You selected",
input$letter))
}

shinyApp(
  ui = fluidPage(q5_module_UI(id = "q5")),
  server = function(input, output, session) {
    callModule(module = q5_module, id = "q5")
  }
)

####################
#                  #
#    Exercise 6    #
#                  #
####################
shinyApp(
  ui = fluidPage(
    q5_module_UI(id = "q5_a"), br(),
    q5_module_UI(id = "q5_b")
  ),
  server = function(input, output, session) {
    callModule(module = q5_module, id = "q5_a")
```

```
    callModule(module = q5_module, id = "q5_b")
  }
)

####################
#                  #
#    Exercise 7    #
#                  #
####################
q7_module_UI <- function(id, label = "Select a letter") {
  ns <- NS(id)
  tagList(
    selectInput(
      inputId = ns("letter"),
      label = label,
      choices = LETTERS
    ),
    textOutput(outputId = ns("selected_letter"))
  )
}

q7_module <- function(input, output, session) {
   output$selected_letter <- renderText(paste("You selected",
input$letter))
}

shinyApp(
  ui = fluidPage(q7_module_UI(id = "q7", label = "Your letter
goes here")),
  server = function(input, output, session) {
    callModule(module = q7_module, id = "q7")
  }
)

####################
#                  #
#    Exercise 8    #
#                  #
####################
q8_module_UI  <-  function(id,  subjects,  button_label  =
"Submit!")  {
```

```
  ns <- NS(id)
  tagList(
    textInput(inputId = ns("name"), label = "Name"),
     selectInput(inputId = ns("subject"), label = "Subject",
choices = subjects),
    textAreaInput(inputId = ns("message"), label = "Message"),
    actionButton(inputId = ns("submit"), label = button_label)
  )
}

q8_module <- function(input, output, session) {
  observeEvent(input$submit, {
    content <- paste(input$name, input$subject, input$message,
"\n", sep = "\n")
    write(x = content, file = "content.txt", append = T)
  })
}

shinyApp(
  ui = fluidPage(
    q8_module_UI(
      id = "q8",
      subjects = c("Question", "Suggestion"),
      button_label = "Send"
    )
  ),
  server = function(input, output, session) {
    callModule(module = q8_module, id = "q8")
  }
)

####################
#                  #
#    Exercise 9    #
#                  #
####################
q9_module_UI <- function(id) {
  ns <- NS(id)
  tagList(
    numericInput(
      inputId = ns("element_num"),
```

```
      label = "Number of elements",
      value = 3
    ),
    uiOutput(outputId = ns("elements"))
  )
}

q9_module <- function(input, output, session) {
  output$elements <- renderUI({
    tagList(
      lapply(1:input$element_num, function(i) {
         textInput(inputId = paste0("element", i), label =
paste0("Element", i))
      })
    )
  })
}

shinyApp(
  ui = fluidPage(q9_module_UI(id = "q9")),
  server = function(input, output, session) {
    callModule(module = q9_module, id = "q9")
  }
)

####################
#                  #
#    Exercise 10   #
#                  #
####################

q10_module_inner_UI <- function(id) {
  ns <- NS(id)
  textOutput(outputId = ns("inner_text"))
}

q10_module_inner <- function(input, output, session, text) {
  output$inner_text <- renderText(paste("Your text:", text))
}

q10_module_outer_UI <- function(id) {
```

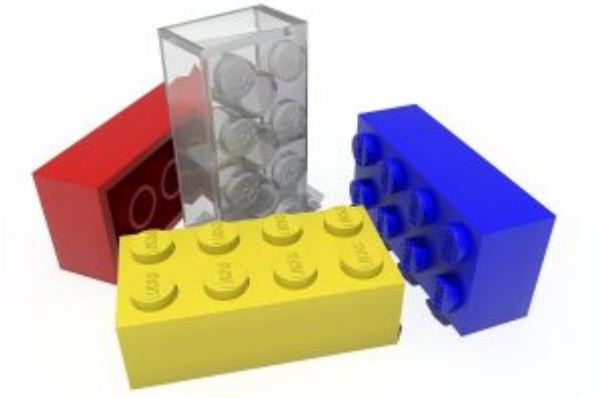
```
  ns <- NS(id)
  q10_module_inner_UI(id = ns("q10_inner"))
}

q10_module_outer <- function(input, output, session, text) {
  callModule(module = q10_module_inner, id = "q10_inner", text
= text)
}

shinyApp(
  ui = fluidPage(q10_module_outer_UI(id = "q10_outer")),
  server = function(input, output, session) {
     callModule(module = q10_module_outer, id = "q10_outer",
text = "Hello")
  }
)
```

[/emaillocker]

Modularize your Shiny Apps: Exercises



Shiny modules are short (well, usually short) server and UI functions, that can be connected to each other by a common namespace, and be embedded within a regular Shiny app. You can’t run a Shiny module without a parent Shiny app. The modules can contain both inputs and outputs, and are usually centered around a single operation or theme.

The biggest advantage of modules is the ability to efficiently reuse Shiny code, which can save a great deal of time. In addition, modules can help you standardize and scale your Shiny operations. Lastly, even if not reused, Shiny modules can help with organizing the code and break it into smaller pieces – which is very much needed in many complex Shiny apps. Some more information on Shiny modules can be found here.

In the following exercise set, you will practice the not-so-straightforward use of Shiny modules. The first four exercises are a warm-up, and will help you "refresh" on how to build each part of a Shiny module. In each of the last six exercises you will build a complete end-to-end module and run a minimal Shiny app to test it. Answers to the exercises are available here.

Two reminders before we begin:
* A typical UI function would take the argument id and start with the line ns <- NS(id).
* All input and output IDs within a UI function should be wrapped with the function like ns().

Exercise 1
Build a module-UI-function that provides a selectInput control, where the choices are LETTERS.

Exercise 2
Build the corresponding module-server-function, that prints the selected letter to the console.

Exercise 3
Build a regular UI object that contains the module-UI-function.

Exercise 4
Build a regular server function that calls the module you built in exercises 1 and 2.
Put together a minimal Shiny app that runs everything (e.g, shinyApp(ui = ui, server = server)).

Exercise 5
Adjust the module you built to show the selecter letter as a UI textOutput instead of printing it to the console.

Learn more about Shiny apps in the online courses Create Interactive Web Applications with the R Shiny Package and R Shiny Interactive Web Apps.

Exercise 6
In some cases you'll need the same module twice in a single Shiny app.
Build a minimal Shiny app that uses the module from exercise 5 twice.

Exercise 7
Still with the same "letters" module, add an option for the app developer to select the label of the selectInput, with the default value being "Select a letter". Call the module with a different value for the label than the default.

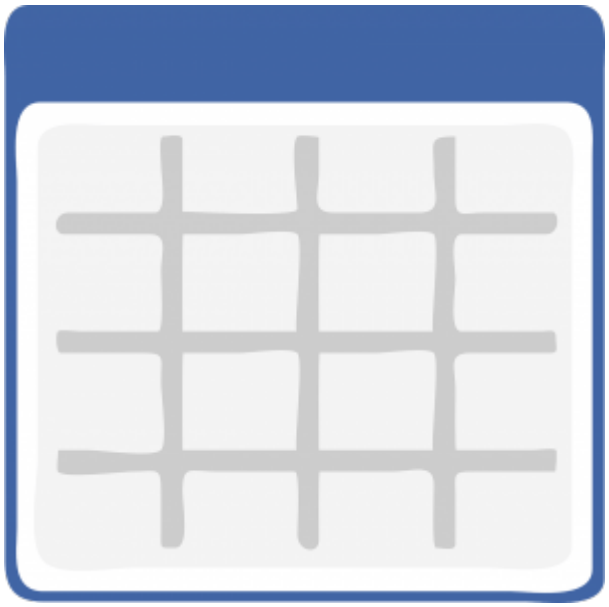
Exercise 8
Build a "contact form" module that contains a name (textInput), a subject (selectInput with dynamic choices, namely the person who uses the module can choose which choices to display) a message (textAreaInput) and a button (actionButton with a dynamic label). Upon clicking the button, all of the form information should be saved in a text file.

Exercise 9
Build a module that takes a number *n* using numericInput, and generates *n* elements of type textInput.

Exercise 10
Modules can also be nested within each other. Namely, one module can call anther module.
To practice that, create two modules, an "inner" one and an "outer" one.
Your minimal Shiny app should call the "outer" module, that in

turn will call the “inner” module.
The module-server-functions should take an argument text, and render it as a textOutput.

Prettify your Shiny Tables with DT: Exercises



Have you ever wanted to make your Shiny tables interactive, more functional and look better? The DT package, which stands for “DataTables”, provides an R interface to the JavaScript library “DataTables”. It allows creating high standard tables by implementing the functionalities and design features that are available through the “DataTables” library.

Even though the DT package can be used independently of Shiny, this exercise set will focus on the integration between the two packages. The exercises were built with a practical orientation in mind, and upon completing them you would be ready to tackle the vast majority of use cases for using DT with Shiny.

We will work with the light built-in dataset datasets::Orange, that holds data about growth of orange trees. Each exercise is adding some more features/functionalities to the code of the previous exercise, so be sure to not discard the code until after you’re done with all of the exercises. Answers to the

exercises are available here.

Each exercises should result in a Shiny app, where the changing part is the renderDataTable() function.
To save time, you can use the following template, and just replace the placeholder for each exercise.

```
ui <- fluidPage(
br(), br(), br(),
fluidRow(column(width = 6, DT::dataTableOutput(outputId = "my_datatable")))
)

server <- function(input, output, session) {
output$my_datatable <-
}

shinyApp(ui = ui, server = server)
```

Exercise 1
Generate a minimal Shiny app that displays the dataset Orange in a datatable (default values).

Exercise 2
Remove the row names.

Exercise 3
Show only 7 rows as the default display, and allow changing the number of displayed rows to either 14, 21, 28 or 35.

Exercise 4
Align the columns text (both values and headers) to the center.

Exercise 5
Remove the search box (top right) and the table information text (bottom left).

Learn more about Shiny apps in the online courses Create Interactive Web Applications with the R Shiny Package and R Shiny Interactive Web Apps.

Exercise 6

Add a "copy" and "csv" buttons to allow saving the table to the clipboard and downloading it as a CSV respectively.

Exercise 7

Add a filter box for each column, at the bottom of the table.

Exercise 8

Allow selection of a single row only, rather than multiple rows (which is the default).

Exercise 9

Remove the option to sort the table.

Exercise 10

Finally, if a row was selected, display in the UI (textOutput) the selected tree and age values.
Hint: for this exercise you would have to make additional changes to the UI and server, rather than just changing the renderDataTable() function.

Prettify your Shiny Tables with DT: Solutions

[emaillocker]Below are the solutions to these exercises on the DT package.

```
# load packages
library(shiny)
library(DT)
```

```
# Each exercises should result in a Shiny app,
# where the changing part is the renderDataTable() function.
# To save time, you can use the following template,
# and just replace the <renderDataTable> placeholder for each
exercise.

# ui <- fluidPage(
#   br(), br(), br(),
#   fluidRow(column(width = 6, DT::dataTableOutput(outputId =
"my_datatable")))
# )
#
# server <- function(input, output, session) {
#   output$my_datatable <- <renderDataTable>
# }
#
# shinyApp(ui = ui, server = server)

####################
#                  #
#    Exercise 1    #
#                  #
####################
DT::renderDataTable(
  expr = datasets::Orange
)

####################
#                  #
#    Exercise 2    #
#                  #
####################
DT::renderDataTable(
  expr = datasets::Orange,
  rownames = F
)

####################
#                  #
#    Exercise 3    #
#                  #
```

```
####################
DT::renderDataTable(
  expr = datasets::Orange,
  rownames = F,
  options = list(
    pageLength = 7,
    lengthMenu = c(7, 14, 21, 28, 35)
  )
)

####################
#                  #
#    Exercise 4    #
#                  #
####################
DT::renderDataTable(
  expr = datasets::Orange,
  rownames = F,
  options = list(
    pageLength = 7,
    lengthMenu = c(7, 14, 21, 28, 35),
    columnDefs = list(list(className = 'dt-center', targets =
0:2))
  )
)

####################
#                  #
#    Exercise 5    #
#                  #
####################
DT::renderDataTable(
  expr = datasets::Orange,
  rownames = F,
  options = list(
    pageLength = 7,
    lengthMenu = c(7, 14, 21, 28, 35),
    columnDefs = list(list(className = 'dt-center', targets =
0:2)),
    dom = "ltp"
  )
```

```
)

####################
#                  #
#    Exercise 6    #
#                  #
####################
DT::renderDataTable(
  expr = datasets::Orange,
  rownames = F,
  options = list(
    pageLength = 7,
    lengthMenu = c(7, 14, 21, 28, 35),
     columnDefs = list(list(className = 'dt-center', targets =
0:2)),
    dom = "ltBp",
    buttons = list("copy", "csv")
  ),
  extensions = c("Buttons")
)

####################
#                  #
#    Exercise 7    #
#                  #
####################
DT::renderDataTable(
  expr = datasets::Orange,
  rownames = F,
  options = list(
    pageLength = 7,
    lengthMenu = c(7, 14, 21, 28, 35),
     columnDefs = list(list(className = 'dt-center', targets =
0:2)),
    dom = "ltBp",
    buttons = list("copy", "csv")
  ),
  extensions = c("Buttons"),
  filter = "bottom"
)
```

```
####################
#                  #
#    Exercise 8    #
#                  #
####################
DT::renderDataTable(
  expr = datasets::Orange,
  rownames = F,
  options = list(
    pageLength = 7,
    lengthMenu = c(7, 14, 21, 28, 35),
     columnDefs = list(list(className = 'dt-center', targets =
0:2)),
    dom = "ltBp",
    buttons = list("copy", "csv")
  ),
  extensions = c("Buttons"),
  filter = "bottom",
  selection = "single"
)

####################
#                  #
#    Exercise 9    #
#                  #
####################
DT::renderDataTable(
  expr = datasets::Orange,
  rownames = F,
  options = list(
    pageLength = 7,
    lengthMenu = c(7, 14, 21, 28, 35),
     columnDefs = list(list(className = 'dt-center', targets =
0:2)),
    dom = "ltBp",
    buttons = list("copy", "csv"),
    ordering = F
  ),
  extensions = c("Buttons"),
  filter = "bottom",
  selection = "single"
```

```
)

####################
#                  #
#    Exercise 10   #
#                  #
####################
ui <- fluidPage(
  br(), br(), br(),
   fluidRow(column(width = 6, DT::dataTableOutput(outputId =
"my_datatable"))),
    fluidRow(column(width = 6, textOutput(outputId =
"selection")))
)

server <- function(input, output, session) {
  output$my_datatable <- DT::renderDataTable(
    expr = datasets::Orange,
    rownames = F,
    options = list(
      pageLength = 7,
      lengthMenu = c(7, 14, 21, 28, 35),
       columnDefs = list(list(className = 'dt-center', targets
= 0:2)),
      dom = "ltBp",
      buttons = list("copy", "csv"),
      ordering = F
    ),
    extensions = c("Buttons"),
    filter = "bottom",
    selection = "single"
  )

  output$selection <- renderText({
    req(input$my_datatable_rows_selected)
                                              tree           <-
datasets::Orange[input$orange_trees_table_rows_selected, 1]
                                               age           <-
datasets::Orange[input$orange_trees_table_rows_selected, 2]
    paste(
      "You selected tree",
```

```
      datasets::Orange[input$my_datatable_rows_selected, 1],
      "and age",
      datasets::Orange[input$my_datatable_rows_selected, 2]
    )
  })
}

shinyApp(ui = ui, server = server)
```

[/emaillocker]

Pull the Right Strings with stringr: Exercises

By providing a set of wrappers to existing functions, the stringr package allows for simple, consistent and efficient manipulations of strings in R. Even though there are some more basic packages that offer strings-related functions, you might find yourself in need for a more complete and straightforward solution for handling strings in R.

With a simple and consistent syntax, stringr provides some very convenient functions around pattern matching, characters manipulation, whitespace handling and more. The full reference of the package can be found here.

Please find below a set of exercises that will help you practice a variety of stringr functions. The focus is on practical operations that data analysts are required to perform on a daily basis. Answers to the exercises are

available here. And, don't forget to check out our other exercise sets on the stringr package by following the stringr tag.

For the following exercises we will use this data:

addresses <- c("14 Pine Street, Los Angeles", "152 Redwood Street, Seattle", "8 Washington Boulevard, New York")

products <- c(“TV “, ” laptop”, “portable charger”, “Wireless Keybord”, ” HeadPhones “)

long_sentences <- stringr::sentences[1:10]

field_names <- c(“order_number”, “order_date”, “customer_email”, “product_title”, “amount”)

employee_skills <- c(“John Bale (Beginner)”, “Rita Murphy (Pro)”, “Chris White (Pro)”, “Sarah Reid (Medium)”)

Exercise 1
Normalize the addresses vector by replacing capitalized letters with lower-case ones.

Exercise 2
Pull only the numeric part of the addresses vector.

Exercise 3
Split the addresses vector into two parts: address and city. The result should be a matrix.

Exercise 4
Now try to split the addresses vector into three parts: house number, street and city. The result should be a matrix.
Hint: use a regex lookbehind assertion

Exercise 5
In the long_sentences vector, for sentences that start with the letter “T” or end with the letter “s”, show the first or last word respectively. If the sentence both starts with a “T”

and ends with an “s”, show both the first and the last words. Remember that the actual last character of a sentence is usually a period.

Learn more about string manipulation with stringr in the online course Learn R by Intensive Practice.

Exercise 6

Show only the first 20 characters of all sentences in the long_sentences vector. To indicate that you removed some characters, use two consecutive periods at the end of each sentence.

Exercise 7

Normalize the products vector by removing all unnecessary whitespaces (both from the start, the end and the middle), and by capitalizing all letters.

Exercise 8

Prepare the field_names for display, by replacing all of the underscore symbols with spaces, and by converting it to the title-case.

Exercise 9

Align all of the field_names to be with equal length, by adding whitespaces to the beginning of the relevant strings.

Exercise 10

In the employee_skills vector, look for employees that are defined as “Pro” or “Medium”. Your output should be a matrix that have the employee name in the first column, and the skill level (without parenthesis) in the second column. Employees that are not qualified should get missing values in both columns.

Pull the Right Strings with stringr: Solutions

[emaillocker]Below are the solutions to these exercises on stringr.

```
# load package
library(stringr)

# prepare the exercises data
addresses <- c("14 Pine Street, Los Angeles", "152 Redwood Street, Seattle", "8 Washington Boulevard, New York")
products <- c("TV   ", " laptop", "portable  charger", "Wireless Keybord", "    HeadPhones    ")
long_sentences <- stringr::sentences[1:10]
field_names   <-   c("order_number",   "order_date", "customer_email", "product_title", "amount")
employee_skills <- c("John Bale (Beginner)", "Rita Murphy (Pro)", "Chris White (Pro)", "Sarah Reid (Medium)")

####################
#                  #
#    Exercise 1    #
#                  #
####################
str_to_lower(string = addresses)

## [1] "14 pine street, los angeles"      "152 redwood street, seattle"
## [3] "8 washington boulevard, new york"

####################
#                  #
#    Exercise 2    #
#                  #
####################
str_extract(string = addresses, pattern = "[:digit:]+")
```

```
## [1] "14"  "152" "8"

####################
#                  #
#    Exercise 3    #
#                  #
####################
str_split(string = addresses, pattern = ", ", simplify = T)

##      [,1]                    [,2]
## [1,] "14 Pine Street"        "Los Angeles"
## [2,] "152 Redwood Street"    "Seattle"
## [3,] "8 Washington Boulevard" "New York"

####################
#                  #
#    Exercise 4    #
#                  #
####################
str_split(string = addresses, pattern = "(?<=[:digit:]) |, ",
simplify = T)

##      [,1]  [,2]                  [,3]
## [1,] "14"  "Pine Street"         "Los Angeles"
## [2,] "152" "Redwood Street"      "Seattle"
## [3,] "8"   "Washington Boulevard" "New York"

####################
#                  #
#    Exercise 5    #
#                  #
####################
str_extract_all(string = long_sentences, pattern = "^T[A-
z]+|[A-z]+s\\.$")

## [[1]]
## [1] "The"     "planks."
##
## [[2]]
## character(0)
```

```
##
## [[3]]
## character(0)
##
## [[4]]
## [1] "These"
##
## [[5]]
## [1] "bowls."
##
## [[6]]
## [1] "The"
##
## [[7]]
## [1] "The"
##
## [[8]]
## [1] "The"
##
## [[9]]
## [1] "us."
##
## [[10]]
## character(0)
```

```
####################
#                  #
#    Exercise 6    #
#                  #
####################
str_trunc(string = long_sentences, width = 22, ellipsis =
"..")
```

```
##  [1] "The birch canoe slid.." "Glue the sheet to th.."
##  [3] "It's easy to tell th.." "These days a chicken.."
##  [5] "Rice is often served.." "The juice of lemons .."
##  [7] "The box was thrown b.." "The hogs were fed ch.."
##  [9] "Four hours of steady.." "Large size in stocki.."
```

```
####################
#                  #
```

```
#    Exercise 7    #
#                  #
####################
str_to_upper(string = str_squish(string = products))
```

```
## [1] "TV"                "LAPTOP"           "PORTABLE
CHARGER"
## [4] "WIRELESS KEYBORD" "HEADPHONES"
```

```
####################
#                  #
#    Exercise 8    #
#                  #
####################
str_to_title(string = str_replace_all(string = field_names,
pattern = "_", replacement = " "))
```

```
## [1] "Order Number"   "Order Date"     "Customer Email"
"Product Title"
## [5] "Amount"
```

```
####################
#                  #
#    Exercise 9    #
#                  #
####################
str_pad(field_names, width = max(str_length(field_names)),
side = "left")
```

```
## [1] "  order_number" "    order_date" "customer_email" "
product_title"
## [5] "       amount"
```

```
####################
#                  #
#    Exercise 10   #
#                  #
####################
str_match(string = employee_skills, pattern = "([A-z ]*)
\\((Pro|Medium)\\)")[, 2:3]
```

```
##      [,1]          [,2]
## [1,] NA            NA
## [2,] "Rita Murphy" "Pro"
## [3,] "Chris White" "Pro"
## [4,] "Sarah Reid"  "Medium"
```

[/emaillocker]

Automating and Scheduling R Scripts in Windows: Tutorial

This tutorial will teach you how to run and schedule R scripts from the command line. Even though parts of this tutorial applies for other operating systems as well, the focus will be on Windows, since it is a bit less straightforward than in other systems.

By the end of this tutorial, you will have the basic knowledge of how to execute operations (including R scripts) from Windows Command Prompts using a single line of code – running complex R scripts, embedding parameters within them and scheduling processes to run repeatedly.

Running R scripts from the command line can have a couple of advantages, such as automating repeating R operations, scaling a large number of R-related processes and simplifying the execution of R scripts. In some cases, you might want a server to run your R script every X hours and in other cases, it might be just more convenient to run an existing script

without the need to access R or RStudio.

Preparations
First, we need to add a specific path as an environment variable in our system.
1. Go to Windows “Search”
2. Type “Edit the system environment variables”
3. Click the button “Environment Variables” (at the bottom)
4. On the bottom pane, under “System variables”, highlight the “Path” variable and click “Edit”.
5. Click “New” and add the path of the “bin” folder of your R software. The path usually looks like: *C:\Program Files\R\R-3.4.4\bin* (it might change a bit between computers or R versions)
6. Click OK in all windows

Notes: Steps 1 and 2 can also be replaced with accessing “Control Panel” -> “System” -> “Advanced”.

Start an R session
Now we are ready to start running scripts from Windows Command Prompt!
Go to Windows “Search” again and type “Command Prompt”.
To run an R session from the command line, simply type: R

If you get the usual R starting message (“R is a free software…”), you’ve done everything right and you can quit the R console for now using the function q(save = "no")
If not, you might have missed something so please go back to the **Preparations** section. If you’re sure you’ve done everything properly and it’s still not working for you, please contact the author of this tutorial.

Now, to run a simple R script from the command line, all you have to do is type:
Rscript path\to\the\script.R
Try it out with a script of your choice!

Pass parameters to your script

To run a script with parameters, you would have to add some code to your R script that will “unpack” the parameters for the script to use. This is how it is done:

```
params <- commandArgs(trailingOnly = TRUE) # notice that params will be a character vector

first_param <- params[1]
second_param <- params[2]
# n_param <- params[n] …

print(first_param)
print(second_param)
```

Now, when you run the script from the command line, you should simply specify the parameters after the path to the script, separated by spaces:
Rscript path\to\the\script.R value_for_the_first_parameter value_for_the_second_parameter

Automate processes by scheduling tasks that run R scripts
The Windows equivalent of the famous cron utility is called “Schtasks”.
The basic syntax for scheduling a task is as follows:
schtasks /create /sc <ScheduleType> /mo <Modifier> /tn <TaskName> /tr <TaskRun>

1. <ScheduleType> can take values like minute, hourly, daily, weekly.
2. <Modifier> can take numerical values to determine the frequency of the task.
3. <TaskName> is simply a string that specifies the name of the task.
4. <TaskRun> is the actual command line code to run repeatedly.

So an R script task will often look like that (this code should go in the command line of course):
schtasks /create /sc minute /mo 30 /tn "My First R Task" /tr

```
"Rscript path\to\the\script.R"
schtasks /create /sc daily /mo 1 /tn "My Second R Task" /tr
"Rscript path\to\the\script_2.R"
```

To delete a task, use the following:

```
schtasks /delete /tn "My First R Script"
```

For more advanced scheduling options, check the full documentation here.

Step Up Your Dashboard With Shinydashboard – Part 2: Solutions

[emaillocker] Below are the solutions to these exercises on the "Shinydashboard Package – Part 2."

```
# load packages
library(shiny)
library(shinydashboard)

####################################
#                                  #
#    Basic Code to Start With      #
#                                  #
####################################
header <- dashboardHeader(
  title = span(
    "Practicing shinydashboard",
    style = " font-weight: bold"
  ),
  titleWidth = "300px"
)
```

```
sidebar <- dashboardSidebar(
  width = "300px",
  sidebarMenu(
    sidebarSearchForm(
      textId = "search_text",
      buttonId = "search_button",
      label = "What are you looking for?"
    ),
    selectInput(
      inputId = "plant",
      label = "Select Plant",
      choices = unique(CO2$Plant)
    ),
    menuItem(
      text = span("Data", style = "font-size: 20px"),
      tabName = "data",
      icon = icon("database"),
      badgeLabel = "New",
      badgeColor = "yellow"
    ),
    menuItem(
      text = span("About", style = "font-size: 20px"),
      tabName = "about",
      icon = icon("info-circle"),
      menuSubItem(text = "Licenses", tabName = "licenses"),
      menuSubItem(text = "Contact Us", tabName = "contact_us")
    )
  )
)

body <- dashboardBody()

ui <- dashboardPage(
  skin = "black",
  title = "R-Exercises",
  header = header,
  sidebar = sidebar,
  body = body
)

server <- function(input, output, session) {}
```

```
shinyApp(ui = ui, server = server)

####################
#                  #
#    Exercise 1    #
#                  #
####################
body <- dashboardBody(
  tabItems(
    tabItem(
      tabName = "data",
      box(
        title = "CO2 Data",
        status = "primary",
        collapsible = T
      )
    )
  )
)

####################
#                  #
#    Exercise 2    #
#                  #
####################
body <- dashboardBody(
  tabItems(
    tabItem(
      tabName = "data",
      box(
        title = "CO2 Data",
        status = "primary",
        collapsible = T,
        tableOutput(outputId = "co2_table")
      )
    )
  )
)

server <- function(input, output, session) {
    output$co2_table  <-  renderTable(CO2[CO2$Plant  ==
```

```
input$plant, ])
}

####################
#                  #
#    Exercise 3    #
#                  #
####################
body <- dashboardBody(
  tabItems(
    tabItem(
      tabName = "data",
      box(
        title = "CO2 Data",
        status = "primary",
        collapsible = T,
        tableOutput(outputId = "co2_table")
      )
    ),
    tabItem(
      tabName = "licenses",
      tabBox(
        tabPanel(title = "Data", "Data licenses..."),
        tabPanel(title = "Icons", "Icons licenses...")
      )
    )
  )
)

####################
#                  #
#    Exercise 4    #
#                  #
####################
body <- dashboardBody(
  tabItems(
    tabItem(
      tabName = "data",
      box(
        title = "CO2 Data",
        status = "primary",
```

```
        collapsible = T,
        tableOutput(outputId = "co2_table")
      )
    ),
    tabItem(
      tabName = "licenses",
      tabBox(
        tabPanel(title = "Data", "Data licenses..."),
        tabPanel(title = "Icons", "Icons licenses...")
      )
    ),
    tabItem(
      tabName = "contact_us",
      fluidRow(
        infoBox(
          title = "Email",
          value = "shiny@dashboard.com",
          subtitle = "(2-3 days to answer)",
          icon = icon("envelope"),
          color = "purple"
        )
      )
    )
  )
)

####################
#                  #
#    Exercise 5    #
#                  #
####################
body <- dashboardBody(
  tabItems(
    tabItem(
      tabName = "data",
      box(
        title = "CO2 Data",
        status = "primary",
        collapsible = T,
        tableOutput(outputId = "co2_table")
      )
```

```
    ),
    tabItem(
      tabName = "licenses",
      tabBox(
        tabPanel(title = "Data", "Data licenses..."),
        tabPanel(title = "Icons", "Icons licenses...")
      )
    ),
    tabItem(
      tabName = "contact_us",
      fluidRow(
        infoBox(
          title = "Email",
          value = "shiny@dashboard.com",
          subtitle = "(2-3 days to answer)",
          icon = icon("envelope"),
          color = "purple"
        )
      ),
      fluidRow(
        valueBox(
          value = 2,
          subtitle = "Average response time (days)",
          icon = icon("thumbs-up"),
          color = "green"
        )
      )
    )
  )
)

####################
#                  #
#    Exercise 6    #
#                  #
####################
header <- dashboardHeader(
  title = span(
    "Practicing shinydashboard",
    style = " font-weight: bold"
  ),
```

```
  titleWidth = "300px",
  dropdownMenu(
    type = "messages",
    badgeStatus = "primary",
    icon = icon("comments"),
     messageItem(from = "Admin", message = "Welcome to my
dashboard!")
  )
)

####################
#                  #
#    Exercise 7    #
#                  #
####################
header <- dashboardHeader(
  title = span(
    "Practicing shinydashboard",
    style = " font-weight: bold"
  ),
  titleWidth = "300px",
  dropdownMenu(
    type = "messages",
    badgeStatus = "primary",
    icon = icon("comments"),
     messageItem(from = "Admin", message = "Welcome to my
dashboard!")
  ),
  dropdownMenu(
    type = "tasks",
    badgeStatus = "success",
    icon = icon("check-square"),
     taskItem(text = "Complete the dashboard", value = 81,
color = "green"),
    taskItem(text = "Fix bugs", value = 45, color = "yellow")
  )
)

####################
#                  #
#    Exercise 8    #
```

```
#                   #
#####################
header <- dashboardHeader(
  title = span(
    "Practicing shinydashboard",
    style = " font-weight: bold"
  ),
  titleWidth = "300px",
  dropdownMenu(
    type = "messages",
    badgeStatus = "primary",
    icon = icon("comments"),
     messageItem(from = "Admin", message = "Welcome to my
dashboard!")
  ),
  dropdownMenu(
    type = "tasks",
    badgeStatus = "success",
    icon = icon("check-square"),
     taskItem(text = "Complete the dashboard", value = 81,
color = "green"),
    taskItem(text = "Fix bugs", value = 45, color = "yellow")
  ),
  dropdownMenuOutput(outputId = "notifications")
)

#####################
#                   #
#    Exercise 9     #
#                   #
#####################
server <- function(input, output, session) {
    output$co2_table  <-  renderTable(CO2[CO2$Plant  ==
input$plant, ])

  output$notifications <- renderMenu(
    dropdownMenu(
      type = "notifications",
      badgeStatus = "warning",
      icon = icon("exclamation-circle"),
      notificationItem(
```

```
        text = paste("You have selected", input$plant),
        status = "warning"
      )
    )
  )
}

####################
#                  #
#    Exercise 10   #
#                  #
####################
header <- dashboardHeader(
  title = span(
    "Practicing shinydashboard",
    style = " font-weight: bold"
  ),
  titleWidth = "300px",
  tags$li(
    class = "dropdown",
    tags$a(
      "Go to R-exercises",
      href = "https://r-exercises.com",
      target = "_blank"
    )
  ),
  dropdownMenu(
    type = "messages",
    badgeStatus = "primary",
    icon = icon("comments"),
     messageItem(from = "Admin", message = "Welcome to my
dashboard!")
  ),
  dropdownMenu(
    type = "tasks",
    badgeStatus = "success",
    icon = icon("check-square"),
     taskItem(text = "Complete the dashboard", value = 81,
color = "green"),
    taskItem(text = "Fix bugs", value = 45, color = "yellow")
  ),
```

```
  dropdownMenuOutput(outputId = "notifications")
)

##################################
#                                #
#    All Exercises Combined      #
#                                #
##################################
header <- dashboardHeader(
  title = span(
    "Practicing shinydashboard",
    style = " font-weight: bold"
  ),
  titleWidth = "300px",
  # exercise 10
  tags$li(
    class = "dropdown",
    tags$a(
      "Go to R-exercises",
      href = "https://r-exercises.com",
      target = "_blank"
    )
  ),
  # exercise 6
  dropdownMenu(
    type = "messages",
    badgeStatus = "primary",
    icon = icon("comments"),
     messageItem(from = "Admin", message = "Welcome to my
dashboard!")
  ),
  # exercise 7
  dropdownMenu(
    type = "tasks",
    badgeStatus = "success",
    icon = icon("check-square"),
     taskItem(text = "Complete the dashboard", value = 81,
color = "green"),
    taskItem(text = "Fix bugs", value = 45, color = "yellow")
  ),
  # exercise 8
```

```
  dropdownMenuOutput(outputId = "notifications")
)

sidebar <- dashboardSidebar(
  width = "300px",
  sidebarMenu(
    sidebarSearchForm(
      textId = "search_text",
      buttonId = "search_button",
      label = "What are you looking for?"
    ),
    selectInput(
      inputId = "plant",
      label = "Select Plant",
      choices = unique(CO2$Plant)
    ),
    menuItem(
      text = span("Data", style = "font-size: 20px"),
      tabName = "data",
      icon = icon("database"),
      badgeLabel = "New",
      badgeColor = "yellow"
    ),
    menuItem(
      text = span("About", style = "font-size: 20px"),
      tabName = "about",
      icon = icon("info-circle"),
      menuSubItem(text = "Licenses", tabName = "licenses"),
      menuSubItem(text = "Contact Us", tabName = "contact_us")
    )
  )
)

body <- dashboardBody(
  tabItems(
    tabItem(
      tabName = "data",
      # exercise 1
      box(
        title = "CO2 Data",
        status = "primary",
```

```
        collapsible = T,
        tableOutput(outputId = "co2_table") # exercise 2
      )
    ),
    tabItem(
      tabName = "licenses",
      # exercise 3
      tabBox(
        tabPanel(title = "Data", "Data licenses..."),
        tabPanel(title = "Icons", "Icons licenses...")
      )
    ),
    tabItem(
      tabName = "contact_us",
      fluidRow(
        # exercise 4
        infoBox(
          title = "Email",
          value = "shiny@dashboard.com",
          subtitle = "(2-3 days to answer)",
          icon = icon("envelope"),
          color = "purple"
        )
      ),
      fluidRow(
        # exercise 5
        valueBox(
          value = 2,
          subtitle = "Average response time (days)",
          icon = icon("thumbs-up"),
          color = "green"
        )
      )
    )
  )
)

ui <- dashboardPage(
  skin = "black",
  title = "R-Exercises",
  header = header,
```

```
  sidebar = sidebar,
  body = body
)

server <- function(input, output, session) {
  # exercise 2
    output$co2_table  <-  renderTable(CO2[CO2$Plant  ==
input$plant, ])

  # exercise 9
  output$notifications <- renderMenu(
    dropdownMenu(
      type = "notifications",
      badgeStatus = "warning",
      icon = icon("exclamation-circle"),
      notificationItem(
        text = paste("You have selected", input$plant),
        status = "warning"
      )
    )
  )
}

shinyApp(ui = ui, server = server)
```

[/emaillocker]

Step Up Your Dashboard With Shinydashboard – Part 2: Exercises

The shinydashboard provides a well-designed dashboard theme for Shiny apps and allows for an easy assembly of a dashboard from a couple of basic building blocks. The package is widely used in commercial environments as well due to its neat features for building convenient and robust layouts.

This exercise set will help you practice all of the main features of this great package. By completing the two parts of the exercise series, you'll know that you're ready to start building well-designed Shiny apps. We will make some minimal use of the built-in data-set datasets::CO2 .(Specific descriptions of the data-set are irrelevant, but you can check them out by typing ?datasets::CO2.) Each exercise is adding some more features/functionalities to the code of the previous exercise, so be sure to not discard the code until after you're done with all of the exercises. Answers to these exercises are available here.

In the solutions page, you'll first find only the relevant component of each exercise. Then, at the end of the page, you will find the entire Shiny app code that contains all of the different components together. This exercise set is based on the output code of this exercise set. If you haven't done it, you can just use the code under "All Exercises Combined" here (at the bottom) as your basis for this exercise set.

For other parts of the series, follow the tag shinydashboard.

Exercise 1
In the "data" tab, add a box() with the title "CO2 Data."
The box should have a blue header and it should be collapsible.

Exercise 2

Add the CO2 table to the box you just created.
The table should be filtered by the plant input that was created in the previous exercise set.

Exercise 3
In the “licenses” tab, add a tabBox() that contains two panels: one titled “Data” and one titled “Icons.”
You can leave those panels empty for now.

Exercise 4
In the “contact_us” tab, add an infoBox() with some content of your choice and select its color and icon.

Exercise 5
In the “contact_us” tab, add an valueBox() with some content of your choice, and select its color and icon.

Learn more about Shiny apps in the online courses Create Interactive Web Applications with the R Shiny Package and R Shiny Interactive Web Apps.

Exercise 6
Add a “messages” drop down menu to the header.
Select its icon and badge-status and add one messageItem() to it.

Exercise 7
Add a “tasks” drop down menu to the header.
Select its icon and badge-status and add two taskItem()s to it.

Exercise 8
Add a placeholder for a “notifications” drop down menu in the header using dropdownMenuOutput().
The output ID should be “notifications.”

Exercise 9
Add the server-side code of the “notifications” drop down menu.

It should have a single notification item that shows what the current value of the plant input that was created in the previous exercise set is.

Exercise 10
Add a link to “r-exercises.com” in the header.
Hint: use tags$li(class = "dropdown", ...) together with tags$a(...).