Lecture 7: Object-Oriented Programming

Problem 1: Create a summary function for the class `pixel`

In R, method dispatch for S3 objects works by calling the method that matches the class of the object. First, we define a simple class `pixel`, then we create a custom `summary` method for this class.

```r
# Define class 'pixel'
pixel <- function(x, y, color) {
  structure(list(x = x, y = y, color = color), class = "pixel")
}

# Define default summary method
summary.default <- function(object) {
  print("No summary available for this object.")
}

# Before implementing summary for 'pixel'
p <- pixel(10, 15, "red")
summary(p)  # Will call the default summary

## [1] "No summary available for this object."

# Define summary method for 'pixel'
summary.pixel <- function(object) {
  cat("Pixel Information: \n")
  cat("X position: ", object$x, "\n")
  cat("Y position: ", object$y, "\n")
  cat("Color: ", object$color, "\n")
}

# After implementing summary for 'pixel'
summary(p)  # Will call the pixel summary method

## Pixel Information:
## X position:  10
## Y position:  15
## Color:  red
```

Problem 2: Difference between `t.test()` and `t.data.frame()`

The function `t.test()` performs a Student's t-test for statistical inference, while `t()` is a generic method that computes the transpose of a matrix or data frame. If you create an object with a custom class, and the class does not have a specific `t()` method, R will fall back to the default method.

Running the code below demonstrates the error due to the absence of a `t.test` method for the custom class:

```r
x <- structure(1:10, class = "test")
t(x)

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    1    2    3    4    5    6    7    8    9    10
## attr(,"class")
## [1] "test"
```

Since `t()` is looking for a method matching the class `test`, but no such method exists, it returns an error.

Problem 3: Understanding `UseMethod()`

In this example, `UseMethod()` is used to dispatch based on the class of the argument. The function `g()` does not modify the class of its arguments, so the default method `g.default()` is called. However, there is a subtle scoping issue in this code where the variable `y` is not available in the scope of `g.default()`.

```r
g <- function(x) {
  x <- 10
  y <- 10
  UseMethod("g")
}

g.default <- function(x) c(x = x, y = y)

x <- 1
y <- 1
g(y)  # Will throw an error because 'y' is not found in g.default's scope

##  x  y
##  1 10
```

To fix this, `y` must be passed as an argument or explicitly declared in the parent environment.

Problem 4: Understanding NextMethod()

In this example, the class of the object is changed within the function `generic2.b`, and `NextMethod()` is used to invoke the next method in the inheritance hierarchy.

```r
generic2 <- function(x) UseMethod("generic2")

generic2.a1 <- function(x) "a1"
generic2.a2 <- function(x) "a2"

generic2.b <- function(x) {
  class(x) <- "a1"
```

```
  NextMethod()
}

generic2(structure(list(), class = c("b", "a2")))

## [1] "a2"
```

When `generic2.b()` is called, it modifies the class of `x` to `"a1"` and calls `NextMethod()`. Even though the class of `x` was modified, the function `NextMethod()` will dispatch to `generic2.a2()`, since the next element in the initial class list was `a2`.The final result will be `"a2"`.

Lecture 8: Web Scraping

Problem 1: CSS Diner

Answers to the CSS Diner exercise:

1. plate
2. bento
3. fancy
4. plate apple
5. fancy pickle
6. apple.small/.small
7. orange.small
8. bento orange.small
9. plate, bento
10. *
11. plate *
12. plate + apple
13. bento + pickle
14. plate > apple
15. orange:first-child
16. plate apple:only-child, plate pickle:only-child
17. apple:last-child, pickle:last-child
18. :nth-child(3)

19. bento:nth-last-child(3)
20. apple:first-of-type
21. :nth-of-type(even)
22. plate:nth-of-type(3n+2)
23. plate apple:only-child
24. apple:last-of-type, orange:last-of-type
25. bento:empty
26. apple:not(.small)
27. [for]
28. plate[for]
29. [for = ”Vitaly”]
30. [for^ =”Sa”]
31. [for$=”ato”]
32. [for*=”obb”].

Problem 2

First follow instructions here. SelectorGadget
Modified code:

```
    library("robotstxt")

# We check that robots.txt allow to scrape from the following
   link

paths_allowed(
  path = "/en/real-estate/rent/city-basel",
  domain = "https://www.immoscout24.ch/"
)

# We can check robots.txt either by this function or visit the
   site directly.
# In this case the function get_robotstxt does not provide the
   correct file.

get_robotstxt(domain = "https://www.immoscout24.ch/")

# Download xml2 package to be able to use read_html function.

library("xml2")
```

```
# Extract html file

real_estate <- read_html(
  "https://www.immoscout24.ch/en/real-estate/rent/city-basel"
)

# To scrape data we need to use another two packages:

library("rvest")
# For using pipe function
library("magrittr")
# Scraping the data
flats <- real_estate %>%
  html_nodes(".HgListingCard_info_RKrwz") %>%
  html_text()
# Printing the obtained text
flats
# Clean data from irrelevante information
flats_df <- data.frame(
  rooms = gsub(pattern = " room.*", "", flats) %>%
    as.numeric(),
# Be careful with "-": do not mix up dash and hyphen! In this
   listing dash should be!
  price = gsub(".*, CHF |.-.*", "", flats) %>%
    gsub(pattern = ",", replacement = "") %>%
    as.numeric()
)
# Print the resulted dataset
flats_df
```

Alternative Workflow using CSS Selectors

Instead of using the *SelectorGadget* tool, you can directly use CSS selectors found in your browser's developer tools. Here's an alternative solution in R using `rvest`:

```
    # 2. Alternative solution using CSS selectors

# To scrape data we use rvest package
library(rvest)
# We read the html file
real_estate <- read_html("https://www.immoscout24.ch/en/real-
   estate/rent/city-basel")
# We extract prices from the nodes "span"
prices <- real_estate %>% html_nodes("span + span") %>% html_text
   ()
# and print the result
prices
# remove the first and two last elements
prices <- prices[-c(1,length(prices)-1,length(prices))]
prices
# Now we extract data regarding the number of rooms and the space
```

```
tmp_m2_rooms <- real_estate %>% html_nodes("strong") %>% html_
   text()
# and print the result
tmp_m2_rooms
# m2 contains information regarding the square of an appartment
m2 <- tmp_m2_rooms[seq(2, length(m2_rooms), 2)]
m2
# rooms corresponds to the number of rooms
rooms <- tmp_m2_rooms[seq(1, length(m2_rooms), 2)]
rooms
# making a nice dataset
flats_df <- data.frame(
  rooms = gsub(pattern = "\\s*rooms", "", rooms) %>%
    as.numeric(),
  meter_square = gsub(" m²","",m2) %>%
    as.numeric(),
  price = gsub("\\s*CHF\\s*", "", prices) %>%
    gsub(pattern = "\\W", replacement = "") %>%
    as.numeric()
)
#printing the resuling dataset
flats_df
```

Problem 3: Web Scraping with `RSelenium` or `chromote`

Using `chromote`:

```
    # 3. Repeat exercise 2. using `RSelenium` or `chromote`.
library(chromote)
b <- ChromoteSession$new()
b$Page$navigate("https://www.immoscout24.ch/en/real-estate/rent/
   city-basel")
tmp_m2_rooms <- b$Runtime$evaluate("document.querySelector('html
   ').outerHTML")$result$value %>%
  read_html() %>%
  html_nodes("strong") %>%
  html_text()
prices <- b$Runtime$evaluate("document.querySelector('html').
   outerHTML")$result$value %>%
  read_html() %>%
  html_nodes("span + span") %>%
  html_text()
# then as in 2.
b$close()
prices
# Note you can use b$screenshot("browser.png") to take a
   screenshot of the browser window.
# Result might differ from the one obtained with rvest because
   the website might have changed or display differently in
   chromote.
```

Problem 4: Extracting World Bank Data using Regular Expressions

```
    # 1. Extract the dataset from the table in https://data.
       worldbank.org/indicator/SP.ADO.TFRT


# The package "Chromote" is used to deal with dynamically
   changing sites
library(chromote)
# To work with html files we should use tidyverse library (or
   rvest)
# In addition tidyverse has a built-in pipe operation
library(tidyverse)

# Initialization of the website, where to scrape
url <- "https://data.worldbank.org/indicator/SP.ADO.TFRT"

# We simulate a Browser to extract raw_data

b <- ChromoteSession$new()

b$Page$navigate(url)
raw_data <- b$Runtime$evaluate("document.querySelector('html').
   outerHTML")$result$value %>%
  read_html() %>%
  html_nodes(".item") %>%
  html_text()
b$close()
# We print the collected raw_data
raw_data


# 2. We modify the view of the dataset to make nicer.
# a. Countrycodes


# We will convert names of countries into their short codes using
# the countryside package
library(countrycode)
# First of all we get rid of any numbers using "[[:digit:]]*"
   regular expression
country <- gsub("[[:digit:]]*", "", raw_data)
# Convert country names to their codes
country_iso <- countrycode(country, origin = 'country.name',
   destination = 'iso3c')
# We print the resulting list of codes
country_iso
# However, country_iso contains NA's. Next we check order numbers
    of non-NA elements
ind <- which(!is.na(country_iso)) # remove missing iso
```

```
# and print the indeces of such counrties
ind


#b. Further modifications with raw_data

# we replace spaces with "a"
raw_data2 <- gsub("\\s","a",raw_data)
# and print the result
raw_data2
# we replace any non-word charcter with "a"
raw_data2 <- gsub("\\W","a",raw_data2)
# and print the result
raw_data2
# we remove all letters and first 4 digital naumbers
data <- gsub("[[:alpha:]]*\\d{4}", "", raw_data2)
# and print the result
data
# and convert elements to number
data <- as.numeric(data)
# We print the result. Some of elements are NA
data


# c. We make a nice dataset

# Using tibble library we create a dataset of a type tibble
library(tibble)
fert_rate <- tibble(ISO = country_iso[ind], value = data[ind])
# and print the resulting datset
fert_rate
```

Lecture 9: Shiny Applications

Problem 1: Extending the Shiny App

We will extend the Shiny app by adding a second tab to display summary statistics of the Old Faithful Geyser dataset.

```
# Define UI for application that draws a histogram
library(shiny)
# To use pipe operator
library(magrittr)

ui <- fluidPage(
  # Application title
  titlePanel("Old Faithful Geyser Data"),
  # Sidebar with a slider input for number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("cells",
```

```
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30),
      textInput(inputId = "label_x",
                label = "Label for the x-axis:"),
      textInput(inputId = "title",
                label = "Title for the graph:"),
      actionButton(inputId = "make_graph",
                   label = "Make the plot!",
                   icon = icon("drafting-compass"))
    ),
    # Show a plot and a table of the generated distribution in
       two tabs
    mainPanel(
      tabsetPanel(
        tabPanel("Plot", plotOutput("distPlot")),
        tabPanel("Summary statisics", tableOutput("tabStats"))
      )
    )
  )
)

server <- function(input, output) {
  # generate cells based on input$cells from ui.R
  x <- reactive(faithful[, 2])
  breaks <- reactive(seq(min(x()), max(x()), length.out = input$
     cells + 1)) %>% bindEvent(input$make_graph)
  xlab <- reactive(input$label_x) %>% bindEvent(input$make_graph)
  title <- reactive(input$title) %>% bindEvent(input$make_graph)
  output$distPlot <- renderPlot({
    # draw the histogram with the specified number of cells
    hist(x(), breaks = breaks(), col = 'darkgray', border = '
       white', xlab=xlab(), main=title())
  })
  output$tabStats <- renderTable({xtable::xtable(t(summary(x())))
     })
}

my_app <- shinyApp(ui = ui, server = server)
runApp(my_app)
```

Solution 2: Reactive Graph Optimization

1. To improve the efficiency, we can first do not make `x` a reactive function, since we do not change it. Therefore, we can initialize x outside the server function.

2. Expression for the plot `plot_data` combines `breaks`, `xlab`, `title` and the plot changes only when we press the button "`make_graph`". There is no need to create several reactive function, instead we can only one directly for $\texttt{plot}_d ata$.

3. There is no need to use `xstable` package to calculate the summary of x.

```
    library(shiny)
    library(magrittr)
    x <- faithful[, 2]
    ui <- fluidPage(
    # Application title
    titlePanel("Old Faithful Geyser Data"),

    # Sidebar with input controls
    sidebarLayout(
        sidebarPanel(
            sliderInput("cells",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30),
            textInput(inputId = "label_x",
                label = "Label for the x-axis:"),
            textInput(inputId = "title",
                label = "Title for the graph:"),
            actionButton(inputId = "make_graph",
                   label = "Make the plot!",
                   icon = icon("drafting-compass")
                   )
            ),
            # Main panel with plot and summary statistics
            mainPanel(
                 tabsetPanel(
                     tabPanel("Plot", plotOutput("distPlot"))
                        ,
                     tabPanel("Summary statistics",
                        tableOutput("tabStats"))
                     )
                )
            )
        )

server <- function(input, output) {
  # Reactive expression to handle all inputs together
  plot_data <- eventReactive(input$make_graph, {
    list(
      breaks = seq(min(x), max(x), length.out = input$cells +
          1),
      xlab = input$label_x,
      title = input$title
    )
  })

  # Render the plot using the grouped reactive data
  output$distPlot <- renderPlot({
    hist(x,
```

```
          breaks = plot_data()$breaks,
          col = 'darkgray',
          border = 'white',
          xlab = plot_data()$xlab,
          main = plot_data()$title)
  })

  # Render the summary statistics table
  output$tabStats <- renderTable(
    {summary(x)}
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

Solution 3: Thematic and Visual Customization

Experiment with different themes in Shiny using the `bslib` package.

```
    library(shiny)
library(magrittr)
library(bslib)
# thematic::thematic_shiny(font = "auto")

# Define UI for application that draws a histogram
ui <- fluidPage(
  theme = bs_theme(bootswatch = "superhero", font_scale = 1.5),

  # Application title
  titlePanel("Old Faithful Geyser Data"),

  # Sidebar with a slider input for number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("cells",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30),
      textInput(inputId = "label_x",
                label = "Label for the x-axis:"),
      textInput(inputId = "title",
                label = "Title for the graph:"),
      actionButton(inputId = "make_graph",
                   label = "Make the plot!",
                   icon = icon("drafting-compass"))
    ),

    # Show a plot of the generated distribution
    mainPanel(
```

```
      tabsetPanel(
        tabPanel("Plot", plotOutput("distPlot")),
        tabPanel("Summary statisics", tableOutput("tabStats"))
      )
    )
  )
)

# Define server logic required to draw a histogram
# server <- function(input, output) {
#   # generate cells based on input$cells from ui.R
#   x <- reactive(faithful[, 2])
#   breaks <- eventReactive(input$make_graph, {seq(min(x()), max(
   x()), length.out = input$cells + 1)})
#   xlab <- eventReactive(input$make_graph, {input$label_x})
#   title <- eventReactive(input$make_graph, {input$title})
#
#   output$distPlot <- renderPlot({
#     # draw the histogram with the specified number of cells
#     hist(x(), breaks = breaks(), col = 'darkgray', border = '
   white', xlab=xlab(), main=title())
#   })
# }

server <- function(input, output) {
  # bs_themer()
  # generate cells based on input$cells from ui.R
  x <- reactive(faithful[, 2])
  breaks <- reactive(seq(min(x()), max(x()), length.out = input$
     cells + 1)) %>% bindEvent(input$make_graph)
  xlab <- reactive(input$label_x) %>% bindEvent(input$make_graph)
  title <- reactive(input$title) %>% bindEvent(input$make_graph)
  observeEvent(input$make_graph, message("make a new graph"))

  output$distPlot <- renderPlot({
    # draw the histogram with the specified number of cells
    hist(x(), breaks = breaks(), col = 'darkgray', border = '
       white', xlab=xlab(), main=title())
  })

  output$tabStats <- renderTable({xtable::xtable(t(summary(x())))
     })
}

# Run the application
shinyApp(ui = ui, server = server)
```