BDM Workshop

Nils Ratnaweera

2023-10-24

Table of contents

Übungsunterlagen

Diese Unterlagen dienen dazu, den Entwicklungsprozess zu veranschaulichen. Sie sind vor allem als Gedankenstütze für mich (Nils) gedacht, um im Unterricht live zu coden.

1 Übung 1

Das fertige "standalone" Skript befindet sich im Ordner `uebung_1_standalone/`, also hier.

1.1 HTML

Starten mit einer leeren html Datei. Diese Datei sowohl in RStudio wie auch in einem Browser öffnen.

```
<!DOCTYPE html>
<html>

</html>
```

`head` und `body` hinzfügen:

```
<!DOCTYPE html>
<html>
+    <head>
+
+    </head>
+
+    <body>+
+
+    </body>

</html>
```

`title` hinzfügen:

```
<!DOCTYPE html>
<html>
    <head>
+        <title>Meine Seite</title>
    </head>
```

```
    <body>

    </body>

</html>
```

Einen `h1` Titel hinzfügen:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Meine Seite</title>
    </head>

    <body>
+        <h1>Meine Seite</h1>
+        <p>Das ist meine Seite</p>
    </body>

</html>
```

1.2 CSS

Neue Datei erstellen und speichern als `style.css`.

HTML elemente selektieren und stylen:

```
body{
    background-color: lightblue;
    font-family: sans-serif;
}
```

Nun müssen wir das css file noch in unsere html Datei einbinden:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Meine Seite</title>
+       <link rel="stylesheet" href="style.css">
```

```
    </head>

    <body>
        <h1>Meine Seite</h1>
        <p>Das ist meine Seite</p>
    </body>

</html>
```

CSS funktioniert mit Selektoren. `body` selektiert im Prinzip die ganze Website. `h1` selektiert alle `h1` Elemente. `p` selektiert alle `p` Elemente.

```
```css
h1{
    text-transform: uppercase;
}
```

Wichtige weitere Selektoren: Selektion via `class` und `id`.

1.3 JavaScript

1.3.1 Basics

JavaScript ist eine Programmiersprache, die im Browser ausgeführt wird. Sie ermöglicht es, die Website dynamisch zu verändern.

Der Syntax ist oftmals ähnlich wie R.

Jeder Browser hat eine integrierte JS Konsole. Auf der offenen Website: F12 drücken und dann auf `Console` klicken.

```
// wie R, aber nicht ganz korrekt
myname = "Nils Ratnaweera"

// mit `let` deklarieren
let myname = "Nils Ratnaweera"

// let ersetzt var
// const ist wie let, aber die Variable kann nicht verändert werden

// Semikolon am Ende jeder Zeile
```
```

```
let myname = "Nils Ratnaweera";

typeof(myname);
```

Arbeiten mit Zahlen:

```
let myage = 2023-1987;

typeof(myage);

let myage_days = myage * 365;

typeof(myage_days);
```

JavaScript ist das letzte Element in userem HTML Dokument, und macht unsere statische Website dynamisch. Erstellt eine neue Datei `script.js` und fügt folgendes ein:

```
let myname = "Nils Ratnaweera";
let myage = 2023-1987;
```

Bindet das JS file in die HTML Datei ein:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Meine Seite</title>
        <link rel="stylesheet" href="style.css">
    </head>

    <body>
        <h1>Meine Seite</h1>
        <p>Das ist meine Seite</p>
+       <script src="script.js"></script>
    </body>

</html>
```

Nun können wir in der Konsole F12 auf die Variablen zugreifen:

```
myname
myage
```

Mit `console.log()` können wir etwas in die Konsole schreiben. In script.js:

```
console.log(myname);
console.log(myage);
```

1.3.2 Optional: JavaScript Funktionen

Wir haben eben unser alter in Tagen berechnet. Das können wir in eine Funktion packen:

```
function age_in_days(year_of_birth, year_today){
    let myage = year_today-year_of_birth;
    let myage_days = myage * 365;
    return myage_days;
}
```

1.4 Mini Webapp

`titel`, `h1` und `p` in der HTML Datei ändern:

```
<!DOCTYPE html>
<html>
    <head>
-       <title>Meine Seite</title>
+       <title>Tagesfortschritt App</title>

        <link rel="stylesheet" href="style.css">
    </head>

    <body>
-      <h1>Meine Seite</h1>
+      <h1>Tagesfortschritt</h1>
-      <p>Das ist meine Seite</p>
+      <p>Wie weit ist der Tag bereits fortgeschritten?</p>
       <script src="script.js"></scrip>
    </body>

</html>
```

Progress bar hinzufügen (note to self: mit value rumspielen. label später hinzufügen):

```
<!DOCTYPE html>
<html>
    <head>
        <title>Meine Seite</title>
        <title>Tagesfortschritt App</title>

        <link rel="stylesheet" href="style.css">
       <script src="script.js"></script>
    </head>

    <body>
        <h1>Tagesfortschritt</h1>
        <p>Wie weit ist der Tag bereits fortgeschritten?</p>

+       <label for="progress" id="percent"></label>
+       <progress id = "progress" value = 0 max = 100></progress>
    </body>

</html>
```

In der Console die Logik für die Berechtung des Tagesfortschritts in Prozent:

```
let now = new Date();
let hours_dec = now.getHours() + now.getMinutes()/60 + now.getSeconds()/3600;
let percent = hours_dec/24*100
```

Nun haben wir den Prozentwert. Diesen code in die `script.js` Datei kopieren.

Nun verändern wir den Progress bar, in dem wir den `value` ändern (folgender Teil in der Konsole entwickeln)

```
// verändert den Progress bar
document.getElementById("progress").value = percent

// verändert das label
document.getElementById("percent").innerHTML = percent


// Label evt. auf 2 Nachkommastellen runden
const percent_text = percent.toFixed(2)+"%"

// verändert das label
```

```
document.getElementById("percent").innerHTML = percent_text
```

Code in das Script file kopieren.

1.4.1 Publishen

- Files in einen Ordner packen. Auf Github.zhaw.ch gehen und ein neues Repository erstellen. Files hochladen und pages aktivieren.
- oder netlify.com / drop

1.4.2 Optional: `setInterval()`

Aktuell müssen wir die Seite immer neu laden, um den Fortschritt zu sehen. Wir können das auch automatisieren. Dazu brauchen wir die Funktion `setInterval()`. Diese Funktion führt eine andere Funktion in einem bestimmten Intervall aus.

```
const now = new Date();
const hours_dec = now.getHours() + now.getMinutes()/60 + now.getSeconds()/3600;
var percent = hours_dec/24*100
document.getElementById("progress").value = percent
const percent_text = percent.toFixed(2)+"%"
document.getElementById("percent").innerHTML = percent_text
```

```
+function update_progress(){
    const now = new Date();
    const hours_dec = now.getHours() + now.getMinutes()/60 + now.getSeconds()/3600;
    var percent = hours_dec/24*100
    document.getElementById("progress").value = percent
-   const percent_text = percent.toFixed(2)+"%"
    // da man sonst kein fortschritt sieht
+   const percent_text = percent.toFixed(4)+"%"
    document.getElementById("percent").innerHTML = percent_text
+}

+setInterval(update_progress, 100)
```

2 Übung 2

In R shiny generieren wir HTML aus R. CSS wird weiterhin in CSS geschrieben, JavaScript ersetzen wir durch R.

Für eine shiny app braucht es im Minimum 3 dinge:

- Ein User Interface in HTML
- Eine "Server Logik" (vorher JS, jetzt eine R Funktion mit 3 Argumenten)
- die Funktion `shinyApp()`, welche das Userinterface und die Server Logik zusammenfügt.

Starten wir, in dem wir ein R file mit diesen Elementen erstellen:

```
library(shiny)

ui <- c()

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

shinyApp(ui, server)
```

Wir können das Script zeile für Zeile ausführen und die verschiedenen Objekte anschauen.

Ohne Userinterface sieht das ganze noch sehr leer aus.

```
library(shiny)

-ui <- c()
+ui <- h1("Tagesfortschritt")

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

shinyApp(ui, server)
```

```
library(shiny)

-ui <- h1("Tagesfortschritt")
```

```
+ui <- tagList(
+   h1("Tagesfortschritt"),
+   p("Wie Weit ist der Tag fortgeschritten?")
+)


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

shinyApp(ui, server)
```

Nun möchten wir, wie vorher, den Tagesfortschritt ermitteln und darstellen. Zuvor hatten wir dies in JS programmiert, nun können wir dies in R programmieren.

```
library(shiny)
library(lubridate)

ui <- tagList(
     h1("Tagesfortschritt"),
     p("Wie Weit ist der Tag fortgeschritten?"),
  )


server <- function(input, output, session){
+  now <- Sys.time()
+
+  percent <- ((hour(now) + minute(now)/60 + second(now)/3600)/24)*100

}

shinyApp(ui, server)
```

Viel eleganter, v.a. um später die übersicht zu behalten, ist es, wenn wir die Berechnung in eine Funktion auslagern.

```
library(shiny)
library(lubridate)

+get_percent <- function(){
+  now <- Sys.time()
+  diff <- ((hour(now) + minute(now)/60 + second(now)/3600)/24)*100
+  return(diff)
```

```
+}

ui <- tagList(
  h1("Tagesfortschritt"),
  p("Wie Weit ist der Tag fortgeschritten?"),
)


server <- function(input, output, session){
-       now <- Sys.time()
-
-       percent <- ((hour(now) + minute(now)/60 + second(now)/3600)/24)*100

+       get_percent()
}


shinyApp(ui, server)
```

Wie krieg ich nun den Prozentwert in das HTML?

Jetzt kommt "reactivity" in's Spiel.

- Reactivity is ungewohnt und sehr gewöhnungsbedüftig
- Reactivity ist ein zentrales Element von Shiny und kann nicht umgangen werden
- Wer Reactivity nicht lernen kann, ohne das "Warum" zu verstehen sollte *unbedingt* mastering-shiny.org/reactive-motivation lesen

```
library(shiny)
library(lubridate)

get_percent <- function(x){
  now <- Sys.time()
  diff <- ((hour(now) + minute(now)/60 + second(now)/3600)/24)*100
  return(diff)
}

ui <- tagList(
  h1("Tagesfortschritt"),
  p("Wie Weit ist der Tag fortgeschritten?"),
+ textOutput("progress")
)
```

```
server <- function(input, output, session){
+  output$progress <- renderText(get_percent())
}


shinyApp(ui, server)
```

Um die App zu aktualisieren, müssen wir die Funktion `invalidateLater()` verwenden.

```
library(shiny)
library(lubridate)

get_percent <- function(x){
  now <- Sys.time()
  diff <- ((hour(now) + minute(now)/60 + second(now)/3600)/24)*100
  return(diff)
}

ui <- tagList(
  h1("Tagesfortschritt"),
  p("Wie Weit ist der Tag fortgeschritten?"),
  textOutput("progress")
)


server <- function(input, output, session){
  output$progress <- renderText({
+   invalidateLater(100)
+   get_percent()
    })
}


shinyApp(ui, server)
```

Wenn wir zu 100% user HTML basiertes App rekonstruieren wollen, müssen wir noch folgenden Schitt machen:

```
library(shiny)
```

```
library(lubridate)

get_percent <- function(x){
  now <- Sys.time()
  diff <- ((hour(now) + minute(now)/60 + second(now)/3600)/24)*100
  return(diff)
}

ui <- tagList(
  h1("Tagesfortschritt"),
  p("Wie Weit ist der Tag fortgeschritten?"),
  textOutput("progress"),
+ uiOutput("progress_bar")
)


server <- function(input, output, session){
  output$progress <- renderText({
   invalidateLater(100)
   get_percent()
    })

+ output$progress_bar <- renderUI({
+
+     invalidateLater(100)
+     percent <- get_percent()
+     tags$progress(id = "progress", value = percent, max = 100)
+
+   })
}


shinyApp(ui, server)
```