Arbeitsauftrag zur selbstständigen Bearbeitung

In Anbetracht der aktuellen Situation werde ich die Kurzinformation, die es bisher im Unterricht gab, so nicht fortsetzen, sondern nun ein etwas ausführlicheres Tutorial schreiben. Dieses sollte ihr dann bitte auch komplett durchlesen und durcharbeiten. Alle Aufgaben müssen schriftlich bzw. am PC mit Bluefish erledigt werden. Eure Ergebnisse speichert ihr dann bitte auf einen USB-Stick ab und bringt diesen dann nach den Osterferien mit in den Unterricht. Dann werden wir eure Ergebnisse vor Ort besprechen. Wie das nach den Ferien mit der Klassenarbeit und dem Projekt genau weitergeht, kann ich noch nicht sagen. Im Übrigen lade ich dieses Skript auch auf meiner Homepage hoch (http://www.joergpohlmann.de).

Vorab eine Kurzwiederholung oder was wir bisher gelernt haben:

- JS in ein HTML-Dokument einbinden
- Variablen deklarieren (3 Arten von Variablendeklaration)
- einfache Rechnungen(Grundrechenarten) mit Variablen oder Zahlen
- Datentypen von Variablen (auch hier grundsätzlich 3 Arten: Zahlen, Texte und Boolean), bisher nur primitive Datentypen
- Ausgabe über ein separates Fenster
- Nutzen der Konsole, um mögliche Fehler zu entdecken
- Erzeugen von Zufallszahlen

Im Folgenden gehe ich davon aus, dass die genannten Inhalte jedeR kann und auch alle Aufgaben des bisherigen Skripts erledigt wurden (falls nicht, dann nachholen). Soviel zum bisher Gelernten.

A) Funktionen

Bisher haben wir unseren Code einfach hintereinander, also Zeile für Zeile, programmiert. Wenn eine Aufgabe aber größer und komplexer wird, benötigen wir so viel Code, dass wir irgendwann den Überblick verlieren. Dann macht es Sinn, die zu programmierende Aufgabe in mehrere kleine Teilaufgaben zu zerlegen. Bei unserem Spiel 17+4 könnte eine Aufgabenteilung wie folgt aussehen:

I. Erzeuge eine Zufallszahl zwischen 1 und 6
II. Wenn der Spieler „Weiter“ anklickt, erzeuge eine neue Zufallszahl und addiere diese zu dem bisherigen Punktestand.
III. Wenn der Spieler „Stopp“ anklickt, zeige den Punktestand in einem neuen Fenster an.
IV. Wenn der Punktestand größer 21 ist, zeige in einem neuen Fenster an: „Du hast verloren.“

Als nächstes müssen wir den Teilaufgaben Namen vergeben, was dann auch der Funktionsname ist. Ich benenne diese Teilaufgaben wie folgt:

I. zufallszahl
II. punktestand
III. spielende
IV. verloren

Funktionsnamen sollten immer klein geschrieben werden[1]. Fangen wir mit der Lösung der ersten Teilaufgabe an. Schaue Dir dazu den Quellcode im Anhang A) an.

Aufgaben:

1. Entferne die drei Punkte in Zeile 8 (Anhang A) und ergänze statt dessen den passenden Quellcode zur Anzeige von Zufallszahlen zwischen 1 und 6 in einem separaten Fenster.
2. Und auch nicht die Kommentare in Deinem Quellcode vergessen.

1 Verpflichtet ist man dazu nicht. Aber es machen halt alle so und insofern halten auch wir uns an diese Konvention.

Wie man an diesem Beispiel sehr gut erkennen kann, liegt der Vorteil von Funktionen vor allem darin, dass man den eigentlichen Quellcode an anderer Stelle (hier im `head`) programmieren kann und dann an der passenden Stelle im HTML-Dokument nur noch den passenden Funktionsnamen aufrufen muss. Natürlich könnten man den Quellcode im `head`-Bereich auch in eine externe Datei (mit der Endung .js) auslagern. Der Zugriff auf diese externe Datei würde dann so funktionieren wie wir das auch schon mit css-Dateien gemacht haben.
Bis hierhin haben wir den Quellcode einfach nur an eine andere Stelle verlagert. Wenn wir eine Funktion dann aufrufen dürfen wir nicht vergessen zu dem Funktionsnamen auch die runden Klammern (vgl. Zeile 17 in Anhang A) zu setzen.
Kurzum: Funktionen mit runden Klammern aufrufen, Variablen ohne Klammern aufrufen.

B) Funktionen mit Rückgabewert

Wenn wir nun zwei Funktionen benutzen, von denen jede einen Wert erzeugt hat und wir dann beide Werte benutzen wollen, so müssen wir eine Möglichkeit schaffen, damit wir auch auf die Werte der beiden Funktionen zugreifen können. Auch das schauen wir uns an einem Beispiel an (vgl. Anhang B). Im `head`-Bereich wurden zwei Funktionen definiert, nämlich die Funktionen `eins()` [Zeilen 7-10] und `zwei()` [Zeilen 11-13][2]. Entscheidend ist hier jeweils die letzte Zeile. Der `return`-Befehl sorgt dafür, dass beim Aufruf der jeweiligen Funktion der errechnete Wert der entsprechenden Variablen zur Verfügung gestellt wird[Zeile 19]. Und natürlich kann man dann mit diesen Werten auch weiter rechnen.
In Zeile 20 findet ihr in der Ausgabe das + Zeichen. Dieses + Zeichen addiert nichts, sondern bedeutet lediglich, dass bei der Ausgabe festgelegte Eingaben (in Anführungszeichen gesetzt) durch den Wert einer Variablen ergänzt werden.

Aufgaben:

3. Fertige eine neue HTML-Datei an, in der nun im `head`-Bereich zwei Funktionen definiert werden.
   a) Die erste Funktion berechnet eine Zufallszahl und gibt den berechneten Wert mittels `return` zurück.
   b) Die zweite Funktion addiert zu einer neuen Variablen das Ergebnis der Zufallszahl dazu, so dass hier die Summe berechnet wird. Auch dieses Ergebnis wird mit `return` zurück gegeben.
   c) Im `body`-Bereich soll dann das Ergebnis der zweiten Funktion übernommen werden und mittels `window.alert` angezeigt werden.

C) Interaktivität

Bisher haben unsere Seiten genau das gemacht, was wir in unseren Code hineingeschrieben haben. Der Anwender hatte allerdings keine Möglichkeit irgendeine Aktion auszuführen. Das werden wir nun mit Hilfe eines Buttons ändern, den der Anwender dann anklicken kann.
Schauen wir uns dazu den Quellcode in Anhang C) an. Zunächst wird in der Funktion `zufall()` eine Zufallszahl berechnet (Vorausgesetzt man ersetzt die Stelle … in Zeile 9 mit dem passenden Code). Das Ergebnis wird der Variablen `zahl` zugeordnet. Außerdem wird dieser Wert zurückgegeben und kann damit an anderer Stelle verwendet werden.
Etwas „tricky“ wird es dann in Zeile 13 des Codes. Weil das Ergebnis des Wurfes nun nicht mehr in einem extra Fenster angezeigt werden soll, sondern direkt in dem HTML-Dokument. Daher nun folgende Aufgaben.

2 Wer mag kann hier auch gerne kreativere Namen für die Funktionen vergeben.

Aufgaben:

4. Erkundige Dich im Internet wofür die jeweiligen Anweisungen im Code aus Anhang) stehen, also wofür stehen insbesondere die Anweisungen: `document`, `getElementById`, `innerHTML` und `addEventListener`. Achte dabei auch auf die vergebenen ID‘s in dem eigentlichen HTML-Dokument.
5. Erläutere jede Zeile des Quellcodes durch eine passende Kommentierung.

Zusatz: HTML und JS: Reihenfolge beachten!

Bevor wir die Interaktivität weiter vertiefen, möchte ich darauf hinweisen, dass der Code in Anhang C) nur deswegen so gut funktioniert, weil ich hier auf die Reihenfolge geachtet habe. Wenn ein Code jedoch immer länger wird, wird es für uns sehr mühselig immer genau darauf zu achten, in welche Zeile wir welchen Befehl schreiben sollten. Das zeugt zwar von sehr guten Programmierkenntnissen und einer hervorragenden Übersicht, ist aber auf Dauer kaum durchzuhalten. Das Problem sollen die Codes in Anhang D) und E) verdeutlichen.

Aufgaben:

6. Führe die beiden Codes nacheinander aus und beobachte den Unterschied.

In den beiden Codes steht in Summe genau der gleiche Code. Allerdings wird der JS-Block in Anhang D) vor dem HTML-Code ausgeführt, während in Anhang E) erst der HTML-Code und dann JS ausgeführt wird. Das Problem ist nun folgendes: Wenn JS auf HTML-Elemente zugreifen möchte, dann müssen diese bereist vorhanden sein. In einem HTML-Dokument wird der Code immer zeilenweise abgearbeitet. In Anhang D) wird also von Zeile 7 -10 erst JS ausgeführt. In Zeile 8 steht nun: mache eine neue Variable (genannt „absatz1“) und ordne dieser den Inhalt mit der Kennzeichnung „bereich1“ in diesem Dokument zu. Allerdings findet die Kennzeichnung erst in Zeile 12 (durch den CSS-Zusatz id=“bereich1“) statt. Kurzum: Da der Browser Zeile für Zeile den Code abarbeitet, sucht der Browser in Zeile 8 vergebens nach der ID „bereich1“, da diese noch nicht bekannt ist. Im Anhang E) ist das anders. Dort wird erst die ID gesetzt und dann greift JS darauf zu. Klingt logisch und ist es auch. Aber wenn wir eine Seite interaktiv gestalten wollen, dann muss JS die ganze Zeit aufpassen, wann etwas geschieht, da der Browser ja nicht wissen kann, wann ein User z.B. einen Button anklickt. Dann bringt die Reihenfolge auch nicht mehr viel. Deswegen bietet JS dafür eine Lösung: die Reihenfolge interessiert nicht und auch wann etwas angeklickt wird ist ebenso egal. Dieses Konzept nennt sich EventListener.

Der EventListener

Wenn man das wortwörtlich übersetzt bedeutet das so viel wie „Ereignis lauschen“. Und genau das macht der EventListener auch, er passt die ganze Zeit auf, ob ein bestimmtes Ereignis stattfindet. Das können wir in Anhang F) sehr gut sehen. Auch hier habe ich mit einer Funktion gearbeitet. Der Reihe nach:

In den Zeilen 8-12 wird die Funktion definiert, die dafür sorgt, dass das JS-Fenster angezeigt wird. (Funktionen können an jeder beliebigen Stelle definiert werden, weil JS beim Aufruf einer Funktion automatisch zu der Zeile springt, in der die Funktion programmiert wurde.)

In Zeile 13 folgt dann der Befehl `window.addEventListener("load",NeuesFenster,false);` Hier wird der EventListener aktiviert. Und zwar wird vorne angestellt, worauf er achten soll, hier also das Fenster. In Klammern folgt dann was zu tun ist, nämlich erst den kompletten HTML-Code zu laden („load“), danach wird eine Funktion aufgerufen[3]. Der letzte Eintrag false bedeutet nur, dass bei sehr alten Browsern alles ignoriert werden soll und nur der HTML-Code geladen werden soll. Dieser Zusatz kann auch entfallen, wir leben ja im Jahr 2020 und nicht 2010, wo das noch nötig war.

3 Achtung: Beim EventListener (und EventHandler) werden Funktionsnamen ohne Klammern geschrieben -sofern keine Werte übergeben werden- während an anderer Stelle immer nach dem Funktionsnamen die Klammern beim Aufruf gesetzt werden müssen.

Aufgaben:

7. Benenne zwei weitere Ereignisse, auf die der EventListener lauschen kann und erläutere diese kurz.
8. Programmiere einen Code, bei dem eines Deiner in Aufgabe 7 genannten Ereignisse umgesetzt wird.
9. Programmiere einen Quellcode, bei dem in Deinem Dokument durch den Klick auf einen Button zusätzlicher Text angezeigt wird. (Das passende HTML-Tag lautet <button>...</button>.) Die Umsetzung darf nicht mit onclick erfolgen, sondern muss mit dem EventListener geschehen.

Mit JS können wir auch in das HTML-Dokument eingreifen. Deshalb nun eine Aufgabe zum selbstständigen Recherchieren.

Programmiere mit Hilfe von JS folgende Dokumente:

10. Vergib für einen Textabschnitt eine id. Dieser Textabschnitt soll dann mittels JS die Text-Farbe rot bekommen.
11. Wie in Aufgabe 10. Nun kommt aber ein Button mit dazu. Und erst wenn man auf den Button klickt ändert sich die Textfarbe.

Eure Ergebnisse zu den bisher noch nicht erledigten Aufgaben bitte bis 02.05.2020 an meine Mail-Adresse.

Wenn es Fragen gibt einfach kurz eine Mail an joerg.pohlmann@gymga.de

Herzliche Grüße vom

Jörg Pohlmann

Anhang:

A)

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.  <meta charset="utf-8" />
5.  <title>Funktionen Teil 1</title>
6.  <script type="text/javascript">
7.  function zufallszahl() {
8.  ...
9.  }
10. </script>
11. </head>
12. <body>
13. <h1>Arbeiten mit Funktionen Teil 1</h1>
14.
15. <script type="text/javascript">
16. 'use strict'
17. zufallszahl();
18.
19. </script>
20. </body>
21. </html>
```

B)

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.  <meta charset="utf-8" />
5.  <title>Übergabe von Variablen</title>
6.  <script type="text/javascript">
7.  function eins() {
8.  let x=21;
9.  return x;
10. }
11. function zwei() {
12.     return 2;
13. }
14. </script>
15. </head>
16. <body>
17. Und hier wird aus der halben Wahrheit (also 21) die ganze Wahrheit.
18. <script type="text/javascript">
19.     var y=eins()*zwei();
20.     window.alert('Die ganze Wahrheit ist '+y+'!');
21. </script>
22. </body>
23. </html>
```

C)

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.  <meta charset="utf-8" />
5.  <title></title>
6.  <script type="text/javascript">
7.  function zufall() {
8.  let zahl;
9.  zahl ...
10. return zahl;
11. }
12. function zahl1bis6() {
13. document.getElementById('neueZahlAnzeigen').innerHTML = 'Gewürfelt: '+
    zufall();
14. }
15. </script>
16. </head>
17. <body onload="zahl1bis6()">
18. <h1>Zufallszahlen per Klick</h1>
19. Hier berechnen wir zunächst Zufallszahlen, indem wir auf einen Button
    klicken.
20. Dann soll die Zufallszahl zwischen 1 und 6 angegeben werden.
21. <br>
22. <br>
23. <button id="wuerfel">Würfel 1-6</button>
24. <p id="neueZahlAnzeigen"></p>
25. <script type="text/javascript">
26. let schaltflaeche1;
27. schaltflaeche1=document.getElementById('wuerfel');
28. schaltflaeche1.addEventListener('click',zahl1bis6,false);
29. </script>
30. </body>
31. </html>
```

D)

```
1.  <html>
2.  <head>
3.       <meta charset="utf-8" />
4.      <title>Seitentitel</title>
5.  </head>
6.  <body>
7.     <script>
8.        let absatz1 = document.getElementById('bereich1');
9.        window.alert(absatz1.innerHTML);
10.    </script>
11.    <h1>Überschrift 1</h1>
12.    <p id="bereich1">Erster Absatz</p>
13.    <p id="bereich2">Zweiter Absatz</p>
14. </body>
15. </html>
```

E)

```
1.  <html>
2.  <head>
3.      <meta charset="utf-8" />
4.     <title>Seitentitel</title>
5.  </head>
6.  <body>
7.      <h1>Überschrift 1</h1>
8.      <p id="bereich1">Erster Absatz</p>
9.      <p id="bereich2">Zweiter Absatz</p>
10.     <script>
11.         let absatz1 = document.getElementById('bereich1');
12.         window.alert(absatz1.innerHTML);
13.     </script>
14. </body>
15. </html>
```

F)

```
1.  <html>
2.  <head>
3.      <meta charset="utf-8" />
4.       <title>Seitentitel</title>
5.  </head>
6.  <body>
7.  <script>
8.  function NeuesFenster()
9.  {
10.         let absatz1 = document.getElementById('bereich1');
11.         window.alert(absatz1.innerHTML);
12. }
13.         window.addEventListener("load",NeuesFenster,false);
14.     </script>
15.     <h1>Überschrift 1</h1>
16.     <p id="bereich1">Erster Absatz</p>
17.    <p id="bereich2">Zweiter Absatz</p>
18. </body>
19. </html>
```