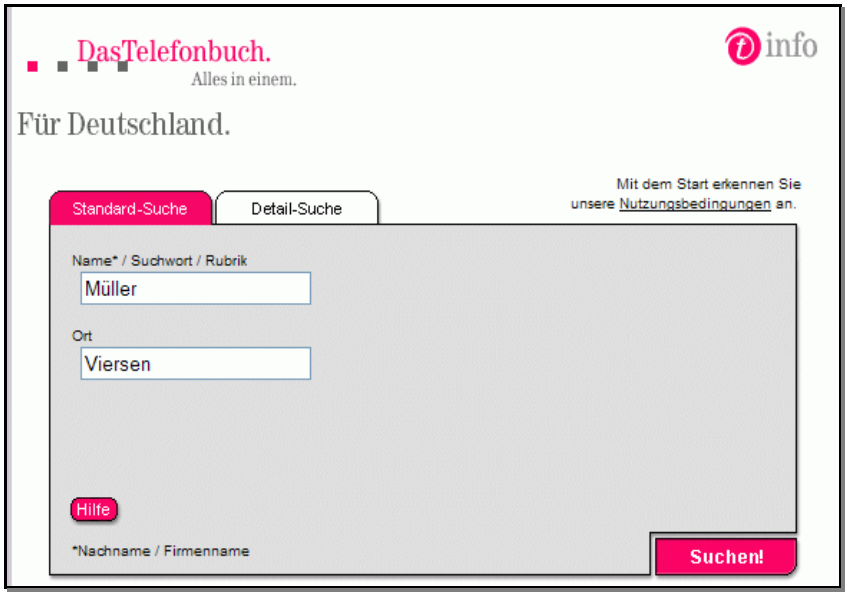

Abb. 1: Das Telefonbuch der Deutschen Telekom im Internet

Wir arbeiten mit Datenbanken

Wenn du eine Telefonnummer in Erfahrung bringen möchtest, kannst du sie in einem Telefonbuch nachschlagen; allerdings findest du in eurem Telefonbuch zuhause nur Personen aus der näheren Umgebung. Anders ist es, wenn du auf eine elektronische Datenbank wie in Abb. 1 zurückgreifst. Hier kannst du die Telefonnummern von vielen Millionen Menschen in Erfahrung bringen.

Ein solches Datenbanksystem hat auch einen weiteren Vorteil. Um eine Telefonnummer zu suchen, brauchst du nur den Namen und die Adresse eintippen; die Telefonnummer wird dann von einem Programm ermittelt.

Wie organisiert man nun eine solche Datenbank mit JavaScript? Und wie programmiert man eine derartige Suchmaschine? Diesen Fragen werden wir in diesem Kapitel nachgehen.

Karteikarten und Datenbanken

Ein Kasten mit Karteikarten wie in Abb. 2 ist eine primitive Form von **Datenbank**. Im Gegensatz zu dem Telefonzettel in Abb. 3 sind hier die Informationen **übersichtlich strukturiert**: Auf jeder Karte steht ein Name mit einer zugehörigen Telefonnummer. Eine solche Karte mit ihren Informationen stellt einen so genannten **Datensatz** dar.

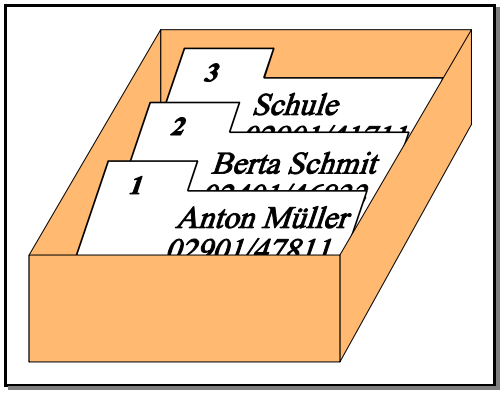

Abb. 2: Karteikarten mit Telefonnummern

Außerdem siehst du in der oberen linken Ecke eine durchlaufende Nummer; sie wird **Datensatznummer** oder auch **Index** genannt. Dieser Index kennzeichnet

den jeweiligen Datensatz, so wie ein Nummernschild dies bei einem Auto tut.

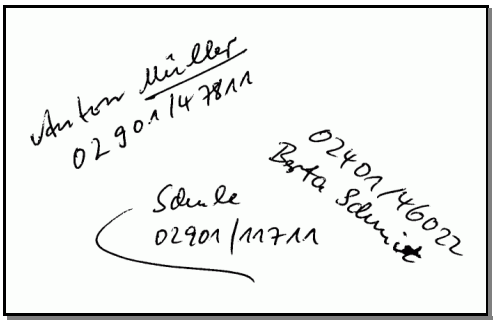

Abb. 3: Telefonnummernzettel

Arrays

Auch in JavaScript gibt es solche Karteikästen; man bezeichnet sie dort als Arrays. **Array** ist ein englisches Wort und bezeichnet auf deutsch so viel wie Anordnung oder Reihe. Hier ist natürlich eine **Reihe von Daten** gemeint.

Wir wollen jetzt eine kleine Telefonliste herstellen. Dazu müssen wir in einem **ersten Schritt** ein solches **Array erzeugen**: Mit dem Befehl

```
var liste = new Array(10)
```

wird z. B. ein Array mit dem Namen `liste` erzeugt. Dieses Array besitzt 10 Elemente. Diese Elemente entsprechen unseren Karteikarten und werden ebenso mit einem Index durchnummeriert; dabei beginnt die Zählung immer bei 0.

Die Karten bzw. Elemente sind jetzt noch leer. In einem **zweiten Schritt** müssen wir sie **mit Inhalten füllen**. Mit der Zuweisung

```
liste[3] = "Arnold Müller 02144/20589"
```

weisen wir z.B. dem Element mit dem Index 3 einen Inhalt zu. Allgemein greift man (sowohl beim Lesen wie auch beim Schreiben) auf ein bestimmtes Element eines Arrays zu, indem man seinen Namen und dahinter in eckigen Klammern seinen Index schreibt.

Natürlich kann statt einer Zahl auch eine Variable als Index eingesetzt werden. Will man z. B. ein Array mit den Quadratzahlen von 0 bis 99 füllen, braucht man nicht 100 einzelne Zuweisungen zu schreiben. Vielmehr wird man eine Zählschleife benutzen:

```
var quadratzahlenliste = new Array(100);
for (var i = 0; i <= 99; i++)
{
     quadratzahlenliste[i] = i*i;
}
```

Oft kennt man bei der Erzeugung des Arrays noch nicht die genaue Anzahl der benötigten Elemente. Dann gibt man als Parameter den Wert 0 oder gar keinen Parameter an:

```
liste = new Array()
```

die einzelnen Elemente werden dann erst bei Bedarf, d. h. bei einem Schreibzugriff erzeugt.

Abb. 4: Diese Telefonliste hat nur wenige Elemente.

Aufgaben

1. Warum ist es nicht sinnvoll, für die einzelnen Elemente einer Telefonliste verschiedene Variablen a, b, c... oder name1, name2, name3... zu benutzen?

2. Entwickle eine Suchmaschine für Telefonnummern ähnlich wie in Abb. 4. Gehe dabei folgendermaßen vor:

2.1 Erstelle das HTML-Grundgerüst und gib die Telefonliste im JavaScript-Abschnitt ein. Warum muss die Telefonliste als globale Variable deklariert werden?

2.2 Wenn man die Schaltfläche „zeigen“ betätigt, soll der zu dem Index gehörige Eintrag im Inhalt-Textfeld angezeigt werden. Teste dein Programm aus. Wie reagiert das Programm, wenn du als Index sinnlose Eingaben wie „17" oder „blabla“ machst?

2.3 Wenn man die Schaltfläche „alle zeigen“ anklickt, sollen in dem daneben liegenden Textbereich alle Datensätze untereinander angezeigt werden wie in Abb. 4. Ergänze dein Dokument entsprechend.

2.4 Wenn die Schaltfläche `merken` betätigt wird, soll der (gegebenenfalls veränderte) Inhalt des Inhalt-Feldes unter dem aktuellen Index gespeichert werden.

Arrays durchsuchen

Für die Telefonliste in Abb. 4 soll eine Suchfunktion programmiert werden. Zunächst benötigen wir eine Funktion, welche überprüft, ob unser Suchbegriff in einer gegebenen Zeichenkette vorkommt oder nicht. Dies leistet die folgende Funktion `gefunden(z, s)`:

```
function gefunden(z, s)
// gibt true zurück, wenn s in
// in der Zeichenkette z gefunden wird, sonst false
{
      var ausschnitt;
      var treffer = false;
      for (var i = 0; i<=z.length-s.length; i++)
      {
            ausschnitt=z.substring(i,i+s.length);
            if (ausschnitt == s)
            {
                  treffer = true;
            }
      }
      return treffer;
}
```

Um zu erklären, wie diese Funktion funktioniert, stellen wir uns vor, dass

```
z = "Liese Meier 02161/43078" und
s = "eier"
```

sein möge. Zunächst wird die Variable `treffer` auf `false` gesetzt; sie soll uns am Ende der Funktion anzeigen, ob die gesuchte Zeichenkette „eier“ gefunden wurde oder nicht. Nun wird eine Schleife durchlaufen: Beim ersten Schleifendurchlauf werden aus der Zeichenkette `z` die ersten 4 Buchstaben, also „Lies“, abgetrennt und in der Variablen `ausschnitt` gespeichert. Diese Zeichenkette wird mit der gesuchten Zeichenkette `s` verglichen. Da die Bedingung „Lies“ == „eier“ nicht erfüllt ist, wird die Variable `treffer` auch nicht auf `true` gesetzt, sie behält den Wert `false` bei.

Beim nächsten Schleifendurchlauf werden aus der Zeichenkette `z` wieder 4 aufeinander folgende Buchstaben herausgeschnitten, diesmal aber mit dem zweiten Buchstaben beginnend (Abb. 5). In der Variablen `ausschnitt` ist jetzt die Zeichenkette „iese“ gespeichert. Wiederum ist die Bedingung `ausschnitt == s` nicht erfüllt und so behält `treffer` den Wert `false`.

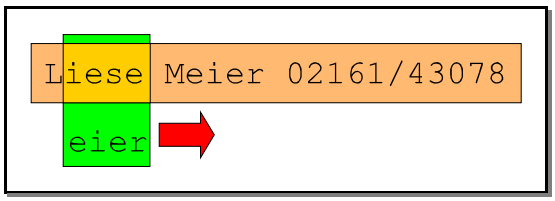

Abb. 5: Unsere Ausschnittschablone durchläuft die gesamte Zeichenkette z.

Auf diese Weise durchläuft die Variable `ausschnitt` alle möglichen Teilketten von `z`, die dieselbe Länge wie die Suchkette `s` haben. Wenn nun mindestens einmal die Inhalte von `ausschnitt` und `s` übereinstimmen, wird die Variable `treffer` auf `true` gesetzt. Dieser Wert

wird am Ende der Funktion zum Rückgabewert. Wenn dagegen `s` nicht in `z` enthalten ist, behält `treffer` den Anfangswert `false` und gibt diesen zurück.

Das Durchsuchen des Arrays ist damit nun vergleichsweise einfach:

```
function nummerSuchen()
{
     var s = lform.suchbegriff.value;
     var z;
     for (var i = 0; i < liste.length; i++)
     {
          z = liste[i];
          if (gefunden(z, s))
          {
               lform.index.value = i;
          }
     }
}
```

Neu ist hier lediglich der Ausdruck **`liste.length`**. Er gibt die Anzahl der Elemente des Arrays `liste` an. In der Schleife werden nacheinander also alle Elemente der Telefonliste durchlaufen und über die Funktion `suche` mit dem Suchbegriff verglichen. Wenn diese Funktion bei einem Element des Arrays den Wert `true` zurückgibt, wird der entsprechende Index im Indexfeld angezeigt.

Aufgaben

1. Wenn die Suchen-Schaltfläche in dem Dokument von Abb. 4 gedrückt wird, soll das Array `liste` nach dem Suchbegriff durchsucht werden. Der Index und der entsprechende Inhalt sollen angezeigt werden.

2. In Aufgabe 1 wird immer nur der letzte Treffer angezeigt. Begründe dies. Ergänze das Dokument um eine Schaltfläche „alle suchen“. Wenn diese Taste betätigt wird, sollen in dem Textbereich alle Treffer angezeigt werden.

3. Die Funktion `gefunden` kann auch wesentlich einfacher mit der `indexOf`-Methode von Zeichenketten realisiert werden. Dazu muss man wissen, dass diese Methode nicht nur nach einzelnen Buchstaben, sondern auch nach ganzen Zeichenketten suchen kann. Für die Zeichenketten

   ```
   z = "Liese Meier 02161/43078" und
   s = "eier"
   ```

   hat `z.indexOf(s)` den Wert 6. Taucht `s` nicht in `z` auf, so ist der Rückgabewert -1. Programmiere nun die `gefunden`-Funktion mithilfe der `indexOf`-Methode.

Abb. 5: telefonliste2.htm

Arrays zur Laufzeit editieren

Bisher konnten Änderungen und Ergänzungen an der Datenbank selbst nur während der Entwicklung des Programms vorgenommen werden. In diesem Abschnitt wollen wir lernen, wie man Einträge in der Telefonliste während der **Laufzeit**, d. h. bei der Darstellung im Browser löscht, verändert oder einfügt. Dazu ergänzen wir zunächst unser HTML-Dokument um die in Abb. 5 dargestellten Schaltflächen „löschen“ und „neu merken“.

Wie man einen bereits bestehenden **Datensatz verändern** kann, wollen wir an einem Beispiel klar machen: Die letzte Ziffer in der Telefonnummer von Herrn Gerting soll korrigiert werden; aus der 4 soll eine 2 werden. Dazu lassen wir zuerst den Datensatz von Herrn Gerting anzeigen; dies können wir über den Datensatzindex 2 oder über unsere „Suchmaschine“ erreichen. Nun korrigieren wir den Inhalt. Als Letztes betätigen wir die Schaltfläche „merken“.

Natürlich funktioniert dies nur, wenn durch die merken-Schaltfläche eine geeignete Funktion aufgerufen wird. Vielleicht hast du schon dieses Problem in eine der obigen Aufgaben bearbeitet. Hier eine Lösung:

```
function merken()
{
      var index = lform.index.value;
      var inhalt = lform.inhalt.value;
      if (index != "")
      {
            liste[index] = inhalt;
            alert(inhalt + " geändert")
      }
}
```

Genau so einfach ist es, einen **neuen Datensatz am Ende des Arrays einzufügen**. Als Index müssen wir nur den größten bis dahin benutzten Index wählen und diesen um 1 erhöhen. Das ist aber gerade der Wert von `liste.length`. Hat nämlich unser Array zur Zeit 17 Elemente, dann hat `liste.length` den Wert 17 und die Elemente unseres Arrays lauten `liste[0]`, `liste[1]`, ..., `liste[16]`. Unser neues Element lautet dann `liste[17]`. Es wird automatisch durch eine Zuweisung der Art

```
liste[17] = ...
```

erzeugt. Wenn man jetzt mit `liste.length` erneut die Anzahl der Elemente unseres Arrays abfragt, erhält man den aktuellen Wert 18.

Die zugehörige Funktion zum Merken eines neuen Datensatzes kann dann so aussehen:

```
function neuMerken()
{
     inhalt = lform.inhalt.value;
     var index = liste.length;
     liste[index] = inhalt;
     lform.index.value = index;
     alert(inhalt+" gemerkt" );
}
```

Deutlich schwieriger ist es, einen **Datensatz zu löschen**. Es reicht nämlich nicht aus, den Inhalt eines Elements zu löschen; es wäre noch da, aber eben ohne Inhalt. Für unseren Karteikasten würde das bedeuten, dass wir den Text auf der Karte nur ausradieren, die Karte aber selbst nicht entfernen würden. Eine unbeschriebene Karte würde zurückbleiben.

Mit einem Doppeltrick kann man dieses Problem lösen: Die folgende Tabelle zeigt, wie der erste Schritt zum Löschen des Datensatzes von Herrn Gerting aussehen muss.

Index	alter Inhalt	neuer Inhalt
0	Bolte, Susanne 02225/89501	Bolte, Susanne 02225/89501
1	Breise, Otto 02167/234567	Breise, Otto 02167/234567
2	Gerting, Walter 09876/20034	Heinrichs, Cordula 23456
3	Heinrichs, Cordula 23456	Heinrichs, Michael 20889
4	Heinrichs, Michael 20889	Lüdermann, Josef 0803/6712
5	Lüdermann, Josef 0803/6712	Meier, Berta 02161/43078
6	Meier, Berta 02161/43078	Meier, Dieter 0367/68401
7	Meier, Dieter 0367/68401	Müller, Anton 02166/20441
8	Müller, Anton 02166/20441	Winter, Maria 07703/55688
9	Winter, Maria 07703/55688	Winter, Maria 07703/55688

In der rechten Spalte taucht Herr Gerting nicht mehr auf. Dies können wir dadurch erreichen, dass wir - im Karteikastenmodell gesprochen - seine Karte mit den Informationen von Frau C. Heinrichs beschreiben; deren Karte wiederum wird mit den Informationen von Herrn M. Heinrichs überschrieben...

Dies erreicht man durch die „Verschiebe“-Befehle

```
liste[2] = liste[3];
liste[3] = liste[4];
...
```

Wenn alle folgenden Namen nachgerückt sind, fehlt am Ende Herrn Gertings Eintrag, allerdings taucht am Ende des Arrays Frau Winter doppelt auf. Dieser letzte Eintrag kann nun – und das ist der zweite Trick – durch die Anweisung

```
liste.length = 9;
```

versteckt werden. Dann gibt es nur noch 9 Elemente und unsere „leere Karte“ ist verschwunden.

Aufgaben

1. Schreibe die Löschfunktion und teste sie aus.
   Hinweis: Lasse die „Verschiebe“-Befehle `liste[i] = liste[j]` mit j = i + 1 in einer Schleife ausführen; denke dabei daran, dass JavaScript das Pluszeichen nicht immer als Rechenzeichen deutet.

2. Teste die Löschfunktionen und auch die beiden Merken-Funktionen aus. Mach auch weniger sinnvolle Eingaben.
   Welche Fehleingaben werden bereits abgefangen? Versuche, weitere Verbesserungen vorzunehmen.

Arrays speichern

Du wirst es sicherlich schon bemerkt haben: Alle Datensätze, die zur Laufzeit erzeugt wurden, werden nicht dauerhaft gespeichert. Spätestens nach einer Aktualisierung des HTML-Dokuments sind sämtliche Veränderungen und Ergänzungen verschwunden.

Leider bietet JavaScript selbst keine Möglichkeit, Daten auf Diskette oder Festplatte zu speichern[1]. Das ist von der JavaScript-Konstrukteuren bewusst so geplant worden; andernfalls könnten nämlich recht einfach und für den Anwender kaum kontrollierbar über das Internet Manipulationen an den Datenbeständen auf diesen Datenträgern vorgenommen werden.

---

[1] In Form so genannter Cookies lassen sich auch mit JavaScript kleine Datenmengen speichern. Wenn du dich dafür interessierst, schau dir die Datei `telefonliste mit cookies.htm` im Verzeichnis `source\datenbanken` an.

Mit einem Workaround, wie wir ihn schon bei unserem Igel-Projekt kennen gelernt haben, gelingt es aber doch. Die Idee ist folgende:

Zunächst trennen wir Datenbank-Programm und Datenbank; so verfährt man übrigens auch bei professionellen Datenbanksystemen. Dazu lagern wir unser Array `liste` in eine externe Datei `telefonliste.js` aus. Beim Start unseres HTML-Dokuments `telefonliste2.htm` werden die Anweisungen dieser Datei importiert. Dies geschieht mit einem zusätzlichen `script`-Tag (Abb. 6); der Name der zu importierenden Datei steht im `src`-Attribut.

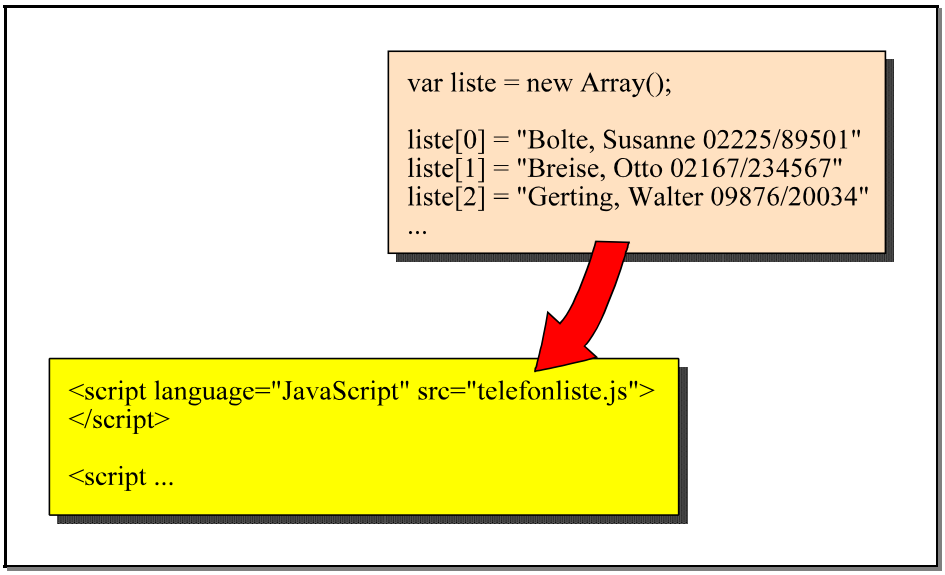

Abb. 6: Import der Array-Daten aus der Datei telefonliste.js

Wie verfahren wir nun beim Speichern? Durch die Schaltfläche „alles speichern“ werden sämtliche Inhalte unseres Arrays auf eine ganz spezielle Art und Weise in den Textbereich geschrieben (Abb. 7). Jetzt brauchen wir den entsprechenden Bereich nur in die Zwischenablage kopieren und in unserer Datei `telefonliste.js` die bestehenden Array-Zuordnungen durch diese zu ersetzen. Mit dem Speichern dieser Datei sind nun alle unsere Datensätze gesichert. Wenn unser HTML-Dokument `telefonliste2.htm` wieder gestartet (oder aktualisiert) wird, werden alle Datensätze wieder importiert und stehen für Abfragen oder weitere Bearbeitungen zur Verfügung.

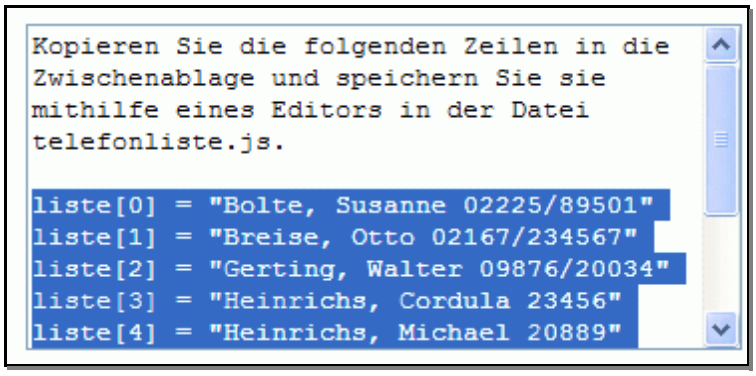

Abb. 7: Das Array ist zum Export vorbereitet.

Aufgaben

1. Füge in unser HTML-Dokument eine Schaltfläche „alles speichern“ ein, so wie es in Abb. 5 zu sehen ist. Schreibe auch die zugehörige Funktion `allesSpeichern()`, welche die Telefonliste wie in Abb. 7 im Textbereich ausgibt.
   Hinweis: Eine Zeichenkette, welche nur aus einem Anführungszeichen besteht, erhält man durch `afz = "\""`.

2. Teste die Funktion `allesSpeichern()` aus.

3. Facharbeit: Informiere dich über einen einfachen Sortieralgorithmus und programmiere ihn. Stelle auch einen weiteren Sortieralgorithmus dar und vergleiche beide.

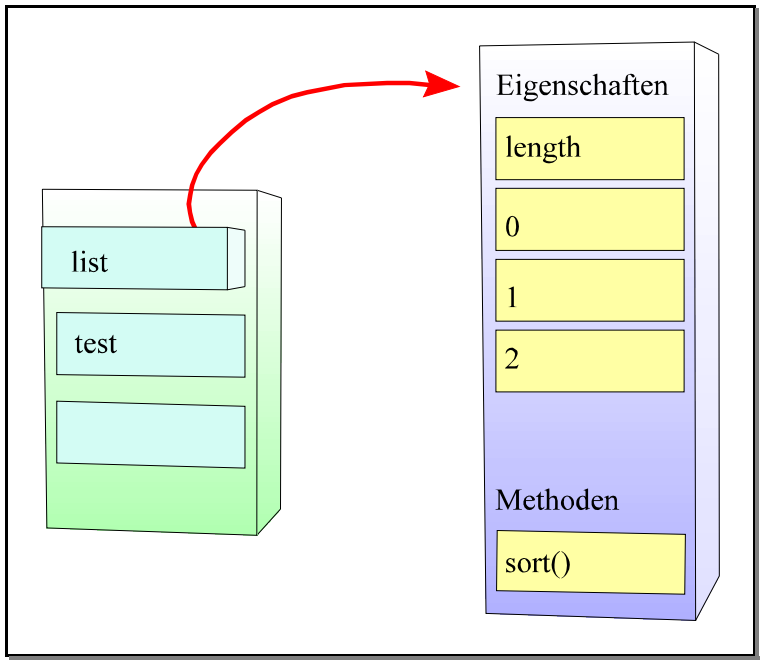

Abb. 8: Ein Array kann man sich als einen Schrank vorstellen.

Arrays als Objekte – ein Blick hinter die Kulissen

Welche Vorgänge im Zusammenhang mit Arrays stattfinden, können wir auch wieder in einem Schubladenmodell klar machen. Schauen wir uns zunächst an, was beim Erzeugen eines Arrays geschieht. Wenn JavaScript auf die Anweisung

```
var liste = new Array(3)
```

stößt, konstruiert es zunächst einmal einen neuen, separaten Schrank mit einer ganzen Reihe von Schubladen oder Fächern (Abb. 8); unter Anderem finden wir hier die einzelnen Elemente unseres Arrays, erkennbar an den Etiketten `0`, `1` und `2`. Anschließend werden Informationen darüber, wo sich dieser Schrank befindet, in einer gewöhnlichen Variablenschublade gespeichert; diese erhält den Namen `liste`. In der Schublade `liste` befinden sich also nicht die Informationen unseres Arrays selbst, sondern nur ein Hinweis darüber, wo sich unser Array befinden. Derartige Variableninhalte bezeichnet man auch als **Zeiger** (englisch: pointer).

Dass diese Vorstellung vom separaten Schrank richtig ist, kann man mit folgenden Zeilen leicht überprüfen:

```
liste = new Array(3);
liste[0] = "Hallo!";
alert(liste[0]);
var test = liste;
test[0] = "Tschüss!"; // Element 0 ändern
alert(test[0]);
alert(liste[0]);
```

Wenn diese Zeilen ausgeführt werden, zeigt die erste Meldung „Hallo!“; anschließend wird zweimal hintereinander die Meldung „Tschüss“ ausgegeben. Wie kann man sich dies erklären? In der 4. Zeile wird der Zeiger aus der Variablen liste in die Variable `test` kopiert. Jetzt verweist nach unserer Vorstellung also nicht nur die Schublade `liste`, sondern auch das Fach `test` auf ein und denselben Array-Schrank. Egal ob wir nun das Element `0` über `liste` oder `test` ansprechen, wir erhalten immer denselben Wert.

Würden hingegen die Informationen des Arrays tatsächlich vollständig in der Schublade `test` liegen, müssten sie durch die Anweisung `var test = liste` komplett in die Schublade test kopiert werden. Sie dürften dann aber durch die Anweisung `test[0] = ...` nicht verändert werden und die letzte Meldung müsste demzufolge „Hallo“ anzeigen.

In den Fächer unseres Arrays finden wir neben den Fächern für unsere Elemente noch weitere Fächer. Eines davon hat das Etikett `length`; in ihm ist die Anzahl der Elemente unseres Arrays gespeichert. In den Element-Schubladen und dem length-Fach sind die Eigenschaften unseres Arrays gespeichert.

Am unteren Ende des Schrankes erkennst du noch eine letzte Schublade mit der Aufschrift `sort()`. In ihr befindet sich eine Folge von Anweisungen, mit denen die Elemente genau dieses (und keines anderen) Arrays alphabetisch sortiert werden. Man nennt solche Schrank-eigenen Funktionen auch Methoden.

Methoden waren dir auch schon einmal bei den Zeichenketten begegnet. In der Tat handelt es sich sowohl bei den Zeichenketten als auch bei den Arrays um **Objekte**. Die Werte, welche diese Objekte charakterisieren, bezeichnet man **Eigenschaften**. Unter den **Methoden** versteht man dann solche Objekt-eigenen Funktionen, welche mit diesen charakteristischen Werten arbeiten.

Allgemein gilt: Die Eigenschaften

```
e1, e2, ...
```

eines Objektes `obj` spricht man an durch

```
obj.e1, obj.e2, ...
```

Ebenso verfährt man bei den Methoden: Die Methoden m1(), m2(), ... eines Objektes obj ruft man auf durch

```
obj.m1(), obj.m2(), ...
```

Natürlich können dieses Methoden auch Parameter besitzen. Dies kennen wir schon von unserem Igel-Objekt.

Etwas seltsam in diesem Zusammenhang ist die Art und Weise, wie die Elemente eines Arrays angesprochen werden. Vielleicht hast du dich gerade auch schon gefragt, warum man sie nicht wie die anderen Eigenschaften mit

```
liste.0, liste.1, ...
```

ansprechen kann. Das liegt sicherlich daran, dass für die Namen von Eigenschaften dieselben Regeln gelten, wie für Variablen; und diese dürfen nie mit einer Zahl beginnen!

In der Tat lassen sich als Index für die Elemente eines Arrays nicht nur Zahlen benutzen; so akzeptiert JavaScript auch die Anweisung

```
liste["nummer1"] = 1234;
```

Dadurch würde eine Elemente-Schublade mit dem Etikett `nummer1` erzeugt. Und diese kann man tatsächlich durch

```
liste.nummer1
```

ansprechen.

Aufgaben

1. Mit `var a = new Array(2)` hast du einen Array-Schrank erzeugt. Jetzt gibst du die Anweisung `a = ""`. Ist der Schrank noch weiter vorhanden? Überprüfe deine Vermutung mit einem kleinen Programm.

2. Bilde ein Array `wort` mit den Elementen: `wort["de"] = "Buch"`, `wort["en"] = "book"` und `wort["fr"] = "livre"`. Teste beide Schreibweisen zum Ansprechen von Elementen aus.

3. Ergänze das HTML-Dokument aus Abb. 5 um die Sortierfunktion.

4. Ein Array ist mit `var a = new Array()` erzeugt worden. Wie sieht das zugehörige Schrank-Modell aus? Was geschieht, wenn die Anweisung

   ```
   a[0] = "Erstes Element"
   ```

   gegeben wird? Welche Änderungen treten auf, wenn anschließend die Anweisung

   ```
   a[2] = "Noch ein Element"
   ```

   erfolgt? Beschreibe im Schrankmodell!

5. Arrays besitzen noch weitere Methoden. Eine davon heißt `reverse()`. Finde heraus, welche Bedeutung sie besitzt.

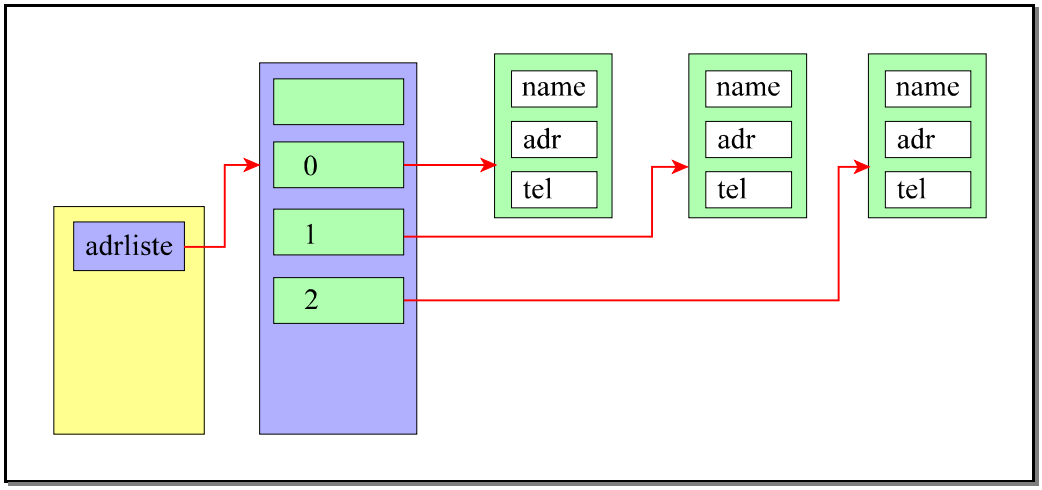

Abb. 9: Die Elemente von Arrays können selbst wieder Arrays sein.

Arrays von Arrays

Wir wollen unsere Telefonliste nun zu einer Adressenliste ausbauen: Neben dem Namen und der Telefonnummer soll nun auch die Anschrift eingerichtet werden. Bislang hatten wir die Informationen eines Datensatzes in einer einzigen Zeichenkette gespeichert. Wenn man nur auf einzelne Teile dieses Datensatzes zugreifen möchte, ist dies unpraktisch: Aufwändige Funktionen mit Zeichenkettenmethoden wären erforderlich.

Deswegen wollen wir nun besser folgendermaßen vorgehen: Jedes Element unserer Adressenliste soll seinerseits aus einem Array bestehen; und jedes dieser Arrays soll 3 Elemente besitzen: Im Element mit dem Index `"name"` soll der Namen gespeichert werden, im Element mit dem Index `"adr"` die Anschrift und im Element mit dem Index `"tel"` die Telefonnummer.

Anhand des Schrankmodells (Abb. 9) überlegen wir, wie dies mit JavaScript realisiert wird. Zuerst erzeugen wir den linken Schrank:

```
var adrliste = new Array(10);
```

Dann erzeugen wir 10 weitere Schränke und weisen ihre Zeiger den Elementen aus dem linken Schrank zu:

```
adrliste[0] = new Array();
adrliste[1] = new Array();
...
```

Diese 10 Anweisungen werden sinnvollerweise durch eine Schleife erledigt:

```
for (var i = 0; i < adrliste.length; i++)
{
     adrliste[i] = new Array();
}
```

Jetzt haben wir unsere gewünschte Datenbankstruktur aus Abb. 9 schon fast vollständig erzeugt. Was fehlt, sind die Schubladen in den rechten Schränken. Diese werden aber automatisch erzeugt, wenn wir die entsprechenden Zuweisungen vornehmen:

```
adrliste[0].name = "Anton Maier";
adrliste[0].adr = "Gartenstr. 7, München";
adrliste[0].tel = "0800/25976";
```

Ebenso der zweite:

```
adrliste[1].name = "Gerda Müller";
adrliste[1].adr = "Feldweg 9, Köln";
adrliste[1].tel = "0222/254443";
```

und die folgenden...

Es ist sinnvoll, wie schon bei der Telefonliste die Erzeugung der Datenbankstruktur und die Zuweisung der einzelnen Daten in einer externen Datei vorzunehmen.

Aufgaben

1. Erstelle Programm und Datenbank zur Verwaltung deiner Adressen. Benutze die in diesem Abschnitt dargestellte Datenbankstruktur aus verschachtelten Arrays. Nach Betätigen der Suchen-Schaltfläche sollen zu dem eingegebenen Namen Anschrift und Telefonnummer ausgegeben werden.

2. Schreibe ein Programm, welches ein kleines Wörterbuch für Deutsch, Englisch und Französisch bietet.

3. Projekt: Vokabellernprogramm (s. Kapitel über Projekte)

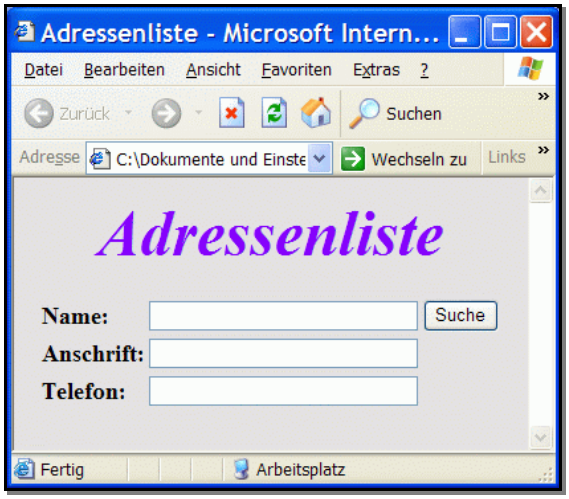

Abb. 10: Zu Aufgabe 1

Formulare und Arrays

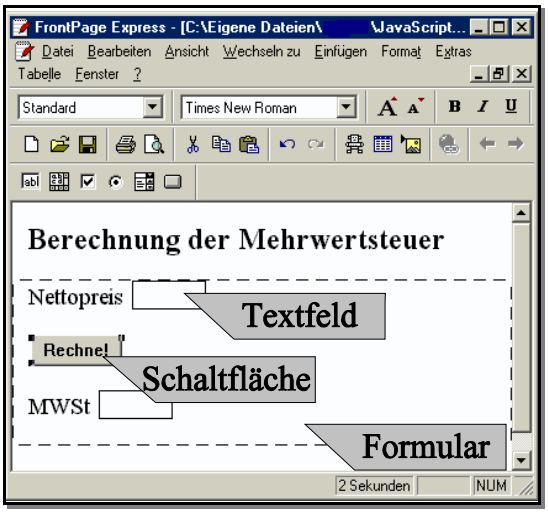

Abb. 11: Formular mit verschiedenen Objekten

Formulare von HTML-Dokumenten sowie die in ihnen enthaltenen Textfelder und Schaltflächen haben wir bisher über ihre Namen angesprochen. Wenn das untere Textfeld in Abb. 11 den Namen `mwst` hat und das Formular die Bezeichnung `mwstform` trägt, dann kann man auf dieses Textfeld von JavaScript aus über

```
mwstform.mwst
```

zugreifen. Es gibt allerdings auch noch eine andere Möglichkeit. Alle Objekte, die sich im Formular befinden, lassen sich nämlich von JavaScript auch als Elemente eines Arrays mit dem Namen `elements` ansprechen. Die Reihenfolge der Objekte in diesem Array ist dabei dieselbe, in der die Objekte im HTML-Dokument auftauchen. Die Zählung beginnt wie üblich mit 0.

JavaScript geht noch einen Schritt weiter: Auch die Formulare selbst werden in einem Array erfasst; dieses hat den Namen `forms`. Es ist eine Eigenschaft des document-Objektes. Da in unserem Fall nur ein einziges Formular vorliegt, erreichen wir es also mit `document.forms[0]`. Statt

```
mwstform.mwst.value
```

können wir also auch genau so gut

```
document.forms[0].elements[2].value
```

schreiben. Es ist also nicht unbedingt erforderlich, den Objekten Namen zu geben; allerdings wird ohne geeignete Namen der Quelltext auch schnell unübersichtlich.

Wie die Formulare so sind auch viele andere Objekte einer Webseite in Arrays zusammengefasst. So befinden sich die Bilder in dem Array `images` und die Links in einem Array `links`.

Aufgaben

1. Peter spricht seine Textfelder und Schaltflächen nicht über Namen, sondern über die entsprechenden Array-Bezeichner an. Welche Probleme können sich ergeben, wenn er das Design seiner Seite ändern möchte?

2. Anne möchte auf Knopfdruck alle Bildinhalte löschen. Wie kann sie dabei vorgehen? Nenne verschiedene Möglichkeiten.