Formulare 2

Jobst-Hartmut Lüddecke

27. Mai 2013

Zusammenfassung

In dieser Lektion wird das Beispiel-Formular der Kreditkarten-Eingabe aus der letzten Lektion um ein Ablauf-Datum der Kredit-Karte erweitert. Dieses Datum wird über 2 *select* Felder als *Pull-Down Menü* abgefragt und als 2. *Schmankerl* mit dem System-Datum verglichen um zu entscheiden, ob die Karte noch gültig, oder bereits abgelaufen ist.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Formulare

1.1 Select

Siehe Beispiel *Erweiterung der Kredit-Karten Abfrage*.

2 System-Datum

2.1 Abfrage

Mit der Funktion *new Date()*[1] kann man einer **eigenen** Zeit-Variablen alle aktuellen Zeit-Informationen zuweisen. Technisch gesehen wird ein *array* vom Typ *Date* deklariert und **gleichzeitig** mit allen **aktuellen** Zeit-Information des Systems gefüllt.

Dies ist die Grundlage für alle weiteren Zeitangaben!

```
Meine-Zeit-Variable = new Date();
```

Dann gibt es diverse Funktionen[2], mit denen man aus dieser **gefüllten Zeit-Variablen** die einzelnen Komponenten der Zeitangabe herausholen kann, um sie einer **eigenen** (deklarierten), weiteren, spezielleren Variablen zuzuweisen. Beispiel:

```
Stunde = Meine-Zeit-Variable.getHours();
```

Alle folgenden Zeit-Funktionen sind entsprechend zu behandeln.

[1] Bitte beachten: Zwischen *new* und *Date* ist ein Leerzeichen
[2] Bitte unbedingt auf große und kleine Schreibweise achten

Die u.a. Liste ist nicht vollständig. Ich halte es für nicht besonders interessant abzufragen, wie viele Millisekunden seit Mitternacht des 1. Januars 1970 GMT vergangen sind. Weiterhin sind die Funktionen zum Setzen der Zeit an dieser Stelle nicht behandelt. Dafür braucht man sowieso die entsprechenden Rechte auf dem betreffenden Rechner.[3].

Auch Funktionen zur Konvertierung des Datums in andere Formate werden an dieser Stelle nicht besprochen. Wer noch tiefer in diese Sache einsteigen will, dem sei *David Flanagan: JavaScript, the Definitive Guide* von *O´Reilly* empfohlen.

2.2 Zeitzone

Mit dieser Funktion erhält man den *Offset* in **Minuten** der örtlichen Zeitzone zur *Greenwich Mean Time* (GMT).

Der Wert ist also durch 60 zu teilen, um den üblichen Verschub der Zeitzonen in Stundeneinheiten zu erhalten.

Die *Mitteleuropäische Zeit* (MEZ oder MET) hat z.B. *+1 Stunde* und die *Mitteleuropäische Sommerzeit* (MESZ) hat *+2 Stunden* Verschub.

Der Wert ist positiv, wenn die Zeitzone östlich GMT liegt (die Sonne geht früher auf) und negativ, wenn die Zeitzone westlich GMT liegt (die Sonne geht später auf). Gegenüber GMT auf dem Globus liegt die Datumsgrenze.

getTimezoneOffset()

2.3 Jahr

Das Jahr der örtliche Zeitzone in 4-stelliger Angabe. Es gibt zwar auch eine Funktion für die 2-stellige Jahresangabe, aber die sparen wir uns lieber nach dem **Y2K-Problem**.[4]

getFullYear()

2.4 Monat

Der Monat der örtlichen Zeitzone als Integer-Wert. **Achtung**: Abweichend von den gewohnten Werten hat der Januar den Wert *0* und Dezember den Wert *11*. Um das gewohnte Format zu bekommen muss man diesen Wert also einmal *inkrementieren*.

Alternativ dazu kann man sich ein Array mit den 12 Monaten in Textform erstellen und den Integer-Wert als Zeiger benutzen.

getMonth()

[3] wenn man schlau ist, gibt man solche Rechte bestimmt nicht auf einer Web-Seite für fremde Benutzer frei!

[4] Das *Jahr-2000-Problem*, dass so viele Probleme bereitet hat, weil viele Programmierer nicht daran gedacht haben, dass nach (19)99 das Jahr (20)00 kam. Oft wurde dann aus dem Jahr 2000 das Jahr 1900 mit vielen Problemen in der Buchhaltung, bei Fristen, bei Dokumentationen, bei Zeitstempel u.v.m.

2.5 Tag des Monats

Der Tag des Monats als gewohnten Integer-Wert zwischen *1* und *31*.

getDate()

2.6 Wochentag

Der Wochentag der örtliche Zeitzone. Was genau dabei rauskommen kann, habe ich noch nicht getestet, da ich bisher noch keine Verwendung dafür hatte. Ich vermute einmal – gestützt durch die Literatur – es ist ein Integer-Wert mit einer **0** für den Sonntag, einer **1** für den Montag usw. *Stefan Münz* hat ein Beispiel in *selfhtml*, bei dem er diesen Wert als Zeiger in einem Array mit den Wochentagen in Textform verwendet.[5]

getDay()

2.7 Stunde

Die Stunde der örtlichen Zeitzone in Integer. Die Werte gehen von *0* für Mitternacht bis *23* für eine Stunde vor Mitternacht.

getHours()

2.8 Minute

Die Minute der örtlichen Zeitzone als Integer von *0* bis *59*.

getMinutes()

2.9 Sekunde

Die Sekunden der örtlichen Zeitzone. Entspricht sonst der Minute.

getSeconds()

2.10 Jahr in UTC

Die *Universal Time* ist die Zeit am Nullmeridian, der durch die Sternwarte *Greenwich* bei London geht. Daher wird diese Zeit auch als *GMT* (Greenwich Mean Time) bezeichnet. Militärisch gesehen wird GMT auch als *Z-Zeit* oder *Zulu-Time* gegenüber der *Alpha-Time* für MEZ bezeichnet. Dies ist auch die Antwort, warum alle Soldaten bei einem NATO-Manöver ihre Uhren bei gültiger MEZ eine Stunde zurück stellen sollen.

Abweichend davon gibt es die *örtlichen Zeitzonen*, die mit einem stündlichen *Offset* belegt sind und die heute im normalen Alltagsleben ungebräuchliche *tatsächliche Ortszeit*. Für Berechnungen in der

[5] siehe http://de.selfhtml.org/javascript/objekte/date.htm

Astronomie, Astrologie[6], oder in der Navigation zur Bestimmung des Längengrades (s.u.) wird sie aber schon noch gebraucht.

Bei der *tatsächlichen Ortszeit* ist es 12:00 Uhr Mittags, wenn die Sonne im Zenit – bei uns genau im Süden[7] – steht. Dies ist, soweit ich mich entsinne in Hamburg 14 Minuten später als in Berlin, für das die *Mitteleuropäische Zeit* gilt. Als *örtliche Zeitzone* ist Hamburg, genauso wie Paris mit Berlin gleich.

Kurz gesagt, *UTC* ist *GMT* ohne Sommerzeit und ähnlichen Blödsinn. Mit dieser Zeit wird auch navigiert und z.B. der aktuelle Längengrad über den Offset zwischen der **tatsächlichen Ortszeit** und **GMT** ermittelt. Daher konnte in der Geschichte der Seefahrt erst mit dem Längengrad navigiert werden, nachdem verlässliche und genau gehende Uhren (Chronographen) verfügbar waren. Ein *Stundenglas* ist hierfür zu ungenau.

getUTCFullYear()

Die folgenden Funktionen für UTC entsprechen in der Verwendung der o.a. Funktionen.

2.11 Monat in UTC

getUTCMonth()

2.12 Tag des Monats in UTC

getUTCDate()

2.13 Wochentag in UTC

getUTCDay()

2.14 Stunden in UTC

getUTCHours()

2.15 Minuten in UTC

getUTCMinutes()

2.16 Sekunden in UTC

getUTCSeconds()

[6] Wenn man z.B. seinen Ascendenten berechnen möchte

[7] Hamburg liegt deutlich nördlich des nördlichen Wendekreises, also des Breitengrades über dem die Sonne am 21. Juni steht. Damit haben wir die Sonne zum Mittag immer im Süden. Befindet man sich zwischen den Wendekreisen, dann braucht man schon einen Kalender. Südlich des südlichen Wendekreises – über dem die Sonne zur Wintersonnenwende steht – befindet sich die Sonne entsprechend im Norden. Ansonsten sind Karte und Kompass, Astronavigation, Deklination, Inklination usw. ein anders Thema. Nur soviel: Es geht auch ohne GPS

3 Erweiterung der Kredit-Karten Abfrage

```
<!DOCTYPE html>
<html lang="de" xml:lang="de">
<head>
<title>Eingabeformular fuer Kreditkarten</title>

<meta http-equiv="content-type" content="text/html;CHARSET=iso8859-2">
<meta http-equiv="expires" content="0">

<meta name="AUTHOR" content="Jobst-Hartmut Lueddecke">
<meta name="description" lang="de"
    content="Beispiel fuer ein Eingabeformular fuer Kreditkarten
             und der Ueberpruefung mit Javascript">
<meta name="keywords"
      content=" Jobst Hartmut Lueddecke Eingabemaske Kreditkarten ">
<meta name="ROBOTS" content="noindex nofollow">
<meta name="LANGUAGE" content="de">
<meta name="COPYRIGHT" content="Jobst-Hartmut Lueddecke">
<meta name="PAGE-TOPIC" content="Beispiel">
<meta name="AUDIENCE" content="all">
<meta name="REVISIT-AFTER" content="10 days">

<meta name="DC.Title"
    content="Beispiel Eingabemaske fuer Kreditkarten">
<meta name="DC.Creator" content="Jobst-Hartmut Lueddecke">
<meta name="DC.Subject" content="Lehrbeispiel">
<meta name="DC.Description"
    content="Beispiel für ein Eingabeformular für Kreditkarten
             und der Überprüfung mit Javascript">
<meta name="DC.Publisher" content="Jobst-Hartmut Lueddecke">
<meta name="DC.Contributor" content="">
<meta name="DC.Date" content="2007-05-09">
<meta name="DC.Type" content="text">
<meta name="DC.Identifier"
      content="karte.html">
<meta name="DC.Source" content="">
<meta name="DC.Language" content="de">
<meta name="DC.Coverage" content="Lehrbeispiel">
<meta name="DC.Rights" content="Alle Rechte liegen beim Autor">

<link rel=stylesheet type="text/css" href="lueddecke.css">
<script language="javascript" src="kartentest.js">
</script>
</head>
```

Abbildung 1: Wie immer der Vorspann.

```
<body>
<h2>Bitte geben sie hier Ihre Kartendaten ein:</h2>
<form name="karte" method="post" onSubmit="return kartentest()">
<input type="radio" name="art" VALUE="a">American Express<br>
<input type="radio" name="art" VALUE="m">Mastercard<br>
<input type="radio" name="art" VALUE="v">Visa<br>
<br>
<table border="0">
<tr>
<td>Kartennummer:</td>
    <td>
        <input type="text" size="4" maxlength="4" name="nummer1">
        <input type="text" size="6" maxlength="6" name="nummer2">
        <input type="text" size="5" maxlength="5" name="nummer3">
        <input type="text" size="4" maxlength="4" name="nummer4">
    </td>
</tr>

<tr>
<td>Kontoinhaber:</td>
<td><input type="text" size="40" name="inhaber"></td>
</tr>

<tr>
<td>Ablaufdatum:</td>
<td><select size="1" name="monat">
    <option value="1">01</option>
    <option value="2">02</option>
    <option value="3">03</option>
    <option value="4">04</option>
    <option value="5">05</option>
    <option value="6">06</option>
    <option value="7">07</option>
    <option value="8">08</option>
    <option value="9">09</option>
    <option value="10">10</option>
    <option value="11">11</option>
    <option value="12">12</option>
    </select>
```

Abbildung 2: Formular, *Radio-Button*, Nummern-Felder, Text-Feld und *Pull-Down* Menü

```
    <select size="1" name="jahr">
    <option value="2009">2009</option>
    <option value="2010">2010</option>
    <option value="2011">2011</option>
    <option value="2012">2012</option>
    <option value="2013">2013</option>
    <option value="2014">2014</option>
    <option value="2015">2015</option>
    <option value="2016">2016</option>
    <option value="2017">2017</option>
    <option value="2018">2018</option>
    <option value="2019">2019</option>
    <option value="2020">2020</option>
    </select>
</td>
</tr>

<tr>
<td>Formular:
</td>
<td><input type="submit" value="abschicken">
    <input type="reset" value="l&ouml;schen">
</td>
</tr>
</table>
</form>
</body>
</html>
```

Abbildung 3: Formular, *Pull-Down* Menü, Sende- und *Reset*-Knopf

```
function kartentest(karte)
{
    var i;

    var kartenart;
    var kartentyp;
    var num1 = document.karte.nummer1.value;
    var num2 = document.karte.nummer2.value;
    var num3 = document.karte.nummer3.value;
    var num4 = document.karte.nummer4.value;
    var nummer_ok = false;
    var karteninhaber = document.karte.inhaber.value;
    var msg;
    var msg_inhaber;
    var monat_ende;
    var jahr_ende;
    var heute;
    var monat_heute;
    var jahr_heute;
    var gueltig;

// Herausfinden welcher Knopf gedrückt (checked) ist und den möglichen
// Wert zuweisen. Es sind im Formular 3 Möglichkeiten für den Namen
// "art" vorgegeben und man kann "art" damit als array mit den Indices
// von [0] bis [2] betrachten. Der Wert steckt für alle im art[i].value
// und die Gültigkeit wird durch das art[i].checked bestimmt.
// Die Obergrenze des arrays steht in document.karte.art.length.

    for(i=0; i<document.karte.art.length; i++)
        {
            if(document.karte.art[i].checked)
                {
                    kartenart = document.karte.art[i].value;
                }
        }

// Gültige Werte sind "a", "m" und "v". Wurde kein Knopf gedrückt,
// gilt "default".

    switch(kartenart)
        {
            case "a":   kartentyp="American Express"; break;
            case "m":   kartentyp="Mastercard"; break;
            case "v":   kartentyp="Visa"; break;
            default:    kartentyp="leer"; return false; break;
        }
```

Abbildung 4: Skript, Verarbeitung *Radio-Button*

```
// Kartennummer überpüfen: Erste Variante
// ======================================
// Die Kartennummer wird in 4 nummerische Felder zerlegt.
// Dies ist einfacher für die Typ- Und Format-Prüfung.
// Der Typ muss jeweils ein Zahlenfeld sein.
// Das Format bei einer Amex-Karte sind 3 Felder mit 4, 6 und
// 5 Ziffern, das 4. Feld ist leer.
// Das Format bei der Visa-Karte und der Mastercard sind 4 Felder
// mit jeweils genau 4 Ziffern.
//
// 1. Fehleingabe: Eines der ersten 3 Zahlenfelder ist leer
// das 4. Feld muss bei der Amex-Karte leer sein.

    if(num1 ==""||num2 ==""||num3 =="")
        {
            alert("Ohne Nummer geht es nicht!");
            return false;
        }

// 2. Fehleingabe: Keine Zahlen! Dazu bietet sich die Funktion
// "isNaN()" für "is not a Number" an. Mit der zu prüfenden
// Variable als Parameter, liefert die Funktion den Wert "true",
// wenn der Inhalt der Variablen "keine Nummer" ist, also "false",
// wenn wir eine Zahl haben.

    if(isNaN(num1)||isNaN(num2)||isNaN(num3))
        {
            alert("Die Kartennummer besteht nur aus Ziffern!");
            return false;
        }
    else
        {
            // Bis hierher ist alles ok, jetzt kommen die
            // Besonderheiten der einzelnen Kartenarten dran

            // Amex-Karte mit 3 Feldern der Länge 4,6,5
            if(kartenart=="a"
                &&num1.length==4
                &&num2.length==6
                &&num3.length==5
                &&num4.length==0)
                {
                    alert("korrektes Format der AmEx Karte");
                    nummer_ok = true;
                }
```

Abbildung 5: Skript *kartentest.js*, Verarbeitung *AmEx*-Nummer

```
            // Visa und Master mit 4 Feldern der Länge 4,4,4,4
            // ausserdem muss das 4. Feld eine Zahl sein.
            if((kartenart=="v" || kartenart=="m")
                &&num1.length==4
                &&num2.length==4
                &&num3.length==4
                &&num4.length==4
                && !(isNaN(num4))
                )
                {
                    alert("korrektes Format der Visa/Master Karte");
                    nummer_ok = true;
                }
        }

// Karteninhaber überprüfen:
// =========================
// 1. Fehlermöglichkeiten "Keine Eingabe" bzw. "leere Eingabe"
// rausfischen

    if(karteninhaber ==""){
        msg_inhaber = "Es geht um Ihre Kreditkarte und nicht um ein ";
        msg_inhaber += "Schweizer Nummernkonto!\n\n";
        msg_inhaber += "Also bitte mit dem Namen des Inhabers ";
        msg_inhaber += "der Karte!\n";
        alert(msg_inhaber);
        return false;
        }

// 2. Zulässige Zeichen sind die Buchstaben "A" bis "Z" oder die
// Zeichen ".", "-", " " (Blank). Alle anderen Zeichen sind ein
// Fehler!
//
// Dazu ist die ganze Eingabe in "document.karte.inhaber.value"
// Zeichen für Zeichen abzuklappern. Die Länge der Eingabe und
// damit die Obergrenze der Schleife steht in
//      "document.karte.inhaber.value.length".
//
// Der eingegebene Text steht in "document.karte.inhaber.value"
// und wie bei jeder Textvariable kann auf die einzelnen Zeichen
// über einen Index zugegriffen werden. Hier
//      "document.karte.inhaber.value[i]"

    for(i=0;i<document.karte.inhaber.value.length;++i){
        test = document.karte.inhaber.value[i];
        if((test >= "A" && test <= "Z") || test == "."
            || test == "-" || test == " ")
                {
                    // Korrektes Namensformat weitertesten
                }
```

Abbildung 6: Skript, Verarbeitung *Visa*-Nummer und Karteninhaber

```
        else{
            // Falsches Namensformat und Abpfiff
            msg = "Es können nur große Buchstaben\n";
            msg += "Bindestriche und Leerzeichen\n";
            msg += "im Feld Kontoinhaber vorkommen\n";
            alert(msg);
            return false;
            } // Ende von else
    } // Ende von for

// Ablaufmonat und Ablaufjahr aus dem Formular rausfischen.
// Die Eingaben selbst brauchen nicht weiter geprüft zu
// werden, da nur vorgegebene Werte selectiert werden können.
//

monat_ende = document.karte.monat.value;
jahr_ende = document.karte.jahr.value;

// Eine weitere sinnvolle Prüfung ist der Abgleich mit
// dem Systemdatum, ob die Karte bereits abgelaufen ist.

// In "heute" holen wir uns jetzt das aktuelle Systemdatum

heute = new Date();

// mit der Funktion "getMonth" holen wir uns jetzt den
// aktuellen Monat aus "heute" raus.
// Achtung, eine Besonderheit, der Monat "Januar" ist
// der Monat "0" im Systemdatum

monat_heute = heute.getMonth();

// mit der Funktion "getFullYear" holen wir uns das
// 4-stellige Jahr aus "heute".

jahr_heute = heute.getFullYear();

// Da beim Systemdatum der Monat Januar der Monat "0"
// ist, zählen wir einen Monat hoch

monat_heute++;

// Kontrollausgabe von Monat und Jahr, ob alles richtig vom
// Betriebssystem geholt wurde. Kann später entfallen.

alert( monat_heute + " / " + jahr_heute );
```

Abbildung 7: Skript, Verarbeitung Ablauf-Datum und Abgleich mit System-Datum

```
// Nun kann es an den eigentlichen Abgleich gehen

// 1. Fall: Das Jahr der Karte ist kleiner als
// das Systemjahr, dann ist die Karte abgelaufen.

if (jahr_ende < jahr_heute){
    gueltig=false;
    alert("Ihre Karte ist abgelaufen");
    return false;
    }

// 2. Fall: Das Jahr der Karte ist größer als
// das Systemjahr, dann ist die Karte noch gültig.

if (jahr_ende > jahr_heute){
    gueltig=true;
    }

// 3. Fall: Das Jahr der Karte ist gleich dem
// Systemjahr, dann müssen wir noch den Monat
// prüfen.

if (jahr_ende == jahr_heute){
    if (monat_ende >= monat_heute){
        gueltig=true;
        }
    else{
        gueltig=false;
        alert("Ihre Karte ist abgelaufen");
        return false;
        }
    }
```

Abbildung 8: Skript, Entscheidung ob Karte abgelaufen ist

```
// Zusammenbau und Ausgabe der Kontrollmeldung
// ==========================================
// Achtung: "nummer_ok" ist eine boolsche Variable
// und die Bedingung ist erfüllt, wenn "nummer_ok"
// den Wert "true" hat.

if(nummer_ok && gueltig)
    {
        msg = "Sie Zahlen mit Ihrer ";
        msg += kartentyp;
        msg += " Karte \n";
        msg += "mit der Nummer: ";
        msg += num1 + " ";
        msg += num2 + " ";
        msg += num3 + " ";
        msg += num4 + " ";
        msg += " \n";
        msg += "Kontoinhaber: ";
        msg += karteninhaber;
        msg +=" \n";
        msg += "Ablaufdatum: ";
        msg += monat_ende + " / " + jahr_ende + "\n\n";
        alert(msg);
    }
else{
        alert("Dies ist keine korrekte Nummer für diesen Kartentyp!");
        return false;
    }
}
```

Abbildung 9: Skript, Ausgabe Kontroll-Meldung

4 Mögliche Lösung der Prüfung einer Mail-Adresse

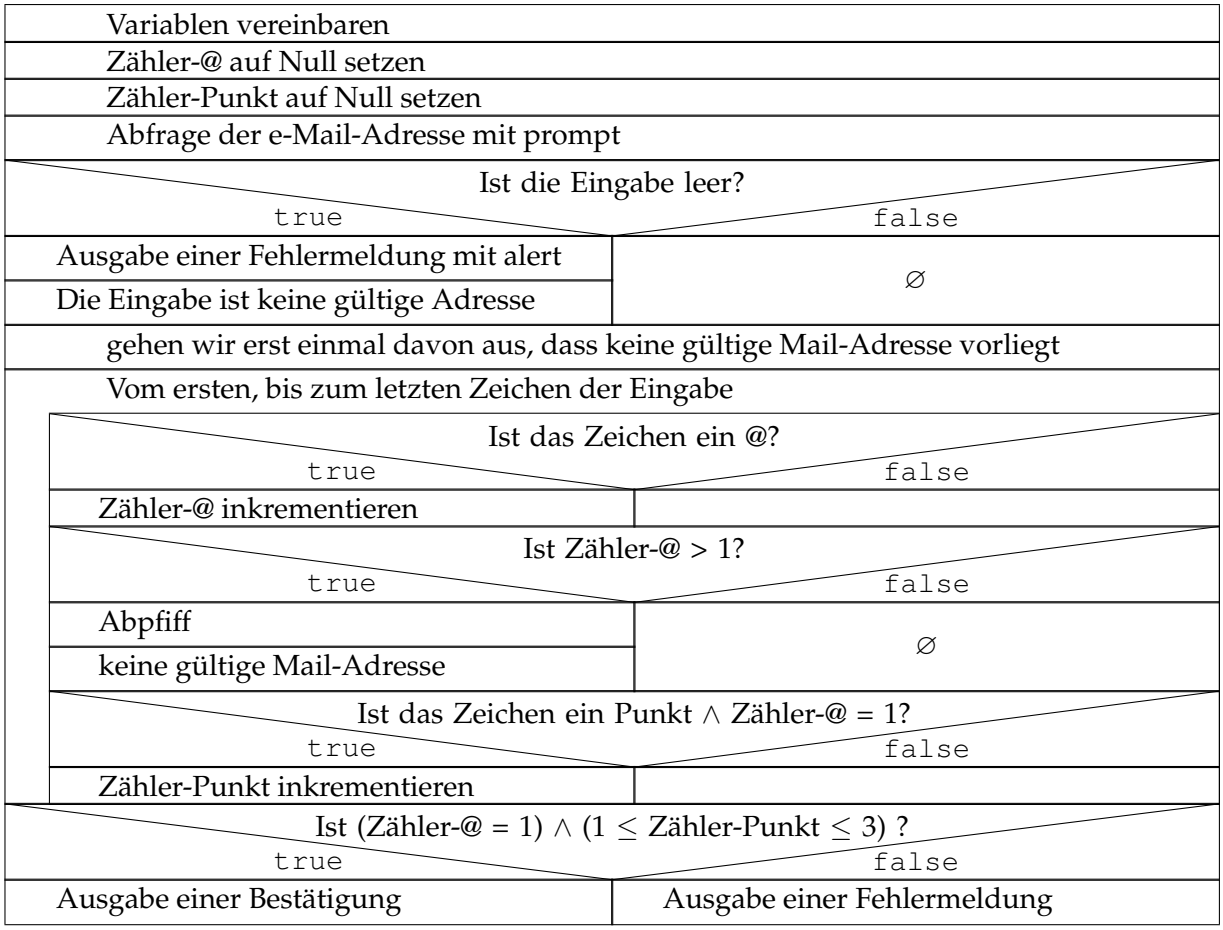

Abbildung 10: e-Mail mit genau einem @ und maximal 3 Punkten nach dem @

In diesem Fall wird die Prüfung der Mail-Adresse dahingehend erweitert, dass es in einer Mail-Adresse **nur genau ein @-Zeichen** geben kann. Nach diesem @ muss die Adresse **mindestens einen Punkt** und darf **maximal 3 Punkte** enthalten. Alle anderen Fälle sind keine gültige Mail-Adresse.

Dieses Beispiel prüft nicht, ob die Adresse nur gültige kleine ASCII-Buchstaben enthält. Die Norm lässt zwar nur kleine Buchstaben in Adressen zu, aber jeder mir bekannte Mailer konvertiert große Buchstaben selbstständig in kleine Buchstaben. Es gibt aber auch in der ASCII-Tabelle noch genug Zeichen, die in einer Mail-Adresse nicht verwendet werden dürfen, z.B. ein *Space*.

Zurück zum Beispiel. Wir benötigen – ähnliche wie beim Beispiel *Vokale zählen* – zwei Variablen in denen wir die @-Zeichen und die Punkte nach dem @-Zeichen zählen. Dann geht es bekannt weiter. Innerhalb einer Schleife wird jedes Zeichen betrachtet. Dabei wird geprüft, ob es ein @-Zeichen ist. Ist es vorhanden, wird der Zähler für dieses Zeichen inkrementiert.

Haben wir mehr als ein @-Zeichen, ist es keine gültige Mail-Adresse. Dies wird in der nächsten if-Abfrage geprüft und entsprechend darauf reagiert.

Da wir die Punkte **nach** dem @-Zeichen zählen wollen, muss die erste Bedingung sein, dass der Zähler der @-Zeichen 1 ist und die zweite, dass ein Punkt als Zeichen vorliegt. Dieser Punkt wird dann in einer Variablen gezählt.

Zum Schluss und nach der Schleife und den Prüfungen wird dann für die Gültigkeit der Mail-Adresse geschaut, ob wir genau ein @-Zeichen und 1 bis 3 Punkte vorliegen haben.

Soweit, so einfach!