ソフトウェア開発
第7回目授業

平野 照比古

2015/11/13

九九の表を作る

- 問題の意図はコンソールにいくつかのデータをまとめて表示する方法を考えること
- HTML 文書で表示したものもあったが、今回のレポートの解答としては正しくない

九九の表を作る (解決法)

1 行分のデータを保存するための変数を用意し、そこに順次必要なデータを付け加える。

```
var i,j, k, res;
for(i=1;i<=9;i++){
  res ="";
  for(j=1;j<=9;j++) {
    k = i*j;
    if(k<10) res +=" ";
    res += k+" ";
  }
  console.log(res);
}
```

解説

- `if(k<10) res +=" ";` の行を次の行 `res += k+" ";` に書いたものがあったが、数の右端がそろわない
- 文字列から部分文字列を得る `substr()` に負の数を与えると右からの部分文字列が得られるので次のように書いてもよい。

```
res += ("␣␣"+i*j).substr(-3)
```

window オブジェクトのプロパティの表示

あまり解説が少ないオブジェクトに対してどのようなプロパティやメソッドがあるかを調べる方法を理解するための問題である。

- 何も表示しないページを作成して、そこで
  `for(p in window)`
  `console.log(p+":"+window[p])`
  の結果に対する考察が必要
- どこかのページでライブラリーを読み込むとその分、余計なものが増える。(`window.a` と変数 `a` が同じものという解説をした)
- ネットにある情報での細かい違いを報告するわけではない。
- 同じことをするためがブラウザによって対応が異なるものに対処する方法については後の授業で解説する。

C 言語の変数名にマッチする

- C 言語の変数名 (正確には識別子) は英字で始まり、そのあとに英数字が並んだもの (正確にはもう少し使える文字がある)
- 先頭の文字は文字クラスを使うと `[A-Za-z]`
- そのあとは英数字
- その文字クラスは `\w`
- 0 個でもよいので、`\w*`
- 全体がこれだけであることを保証するためには位置指定子をつける

```
^[A-Za-z]\w*$
```

浮動小数リテラルをにマッチする正規表現 (1)

浮動小数リテラルは次の部分から成り立っている。
[符号][整数部][小数点][小数部][指数部]
このうち、[符号] や小数点以下の部分はなくてもよい。

浮動小数リテラルをにマッチする正規表現 (2)

- 符号部は+または-からなる一文字からなる。一度だけまで現れてよいので、この部分は [+-]?または (+|-)?で表される。
- 整数部は 10 進数の並びであり最低 1 文字は必要であるので反復の指定は+となる。したがって、この部分は [0-9]+ で表される。
- 小数点 . は正規表現では任意の文字にマッチするのでエスケープする必要がある。したがってこの部分は [\.] となる。
- 小数部は数字が並べられる。全くなくてもよいので反復の指定は*となる。

浮動小数リテラルをにマッチする正規表現 (3)

- 指数部は指数の開始を表す文字 E または e で始まる 10 進数である。数字は最低一つ必要であるのでこの部分は (E|e)\d+となる。
- これらを合わせると求める正規表現が得られる。小数部などがなくてもよいのでそれらの部分には反復指定 ?を付ければよい。

```
^[+-]?\d+(\.\d*)?((E|e)[+-]?\d+)?$
```

浮動小数リテラルをにマッチする正規表現 (4)

正式な数値リテラルでは小数点の前に整数部がない .1 なども許しているが、ここではマッチしない。

24 時間制の時刻の表し方

時、分、秒はすべて 2 桁とし、それらの区切りは:

- 時間は 00 から 23 までであるので時間の初めの文字が 0 と 1 のときと、2 のときで分ける必要がある。
- 時間の先頭が 0 と 1 のときはそのあとの文字は 0 から 9 まで取れるので、[01]\d となる。
- 2 ではじまるときは 0 から 3 まで取れるので、2[0-3] となる。
- 同様に、分と秒は先頭の文字が 0 から 5 までであるので [0-5]\d となる。
- |の範囲を限定するため時間のところの () を忘れないこと。

求めるものは次のとおりである。

```
^([01]\d|2[0-3]):[0-5]\d:[0-5]\d$
^([01]\d|2[0-3])(:[0-5]\d){2}$
```

```
"aaaabaaabb".match(/.*b/);
```

1	2	3	4	5	6	7	8	9	10
a	a	a	a	b	a	a	a	b	b

```
["aaaabaaabb"]
```

- 最後に b が来る文字列がマッチ
- .*は貪欲であるのでできるだけ長い任意の文字列とマッチする
- 最後の b は 10 番目のものが対応

```
"aaaabaaabb".match(/.*b/g);
```

```
["aaaabaaabb"]
```

前と同じ理由により、結果は同じ

```
"aaaabaaabb".match(/.*?b/);
```

1	2	3	4	5	6	7	8	9	10
a	a	a	a	b	a	a	a	b	b

```
["aaaab"]
```

- b で終わる文字列を表している
- その前の.*?は非貪欲なマッチをする
- 最後の b は 5 番目のもの

```
"aaaabaaabb".match(/.*?b/g);
```

1	2	3	4	5	6	7	8	9	10
a	a	a	a	b	a	a	a	b	b

```
["aaaab", "aaab", "b"]
```

- 前と同様に非貪欲なマッチング
- g オプションがついているので一つ目以降見つかった位置から再度マッチするものを探す
- 全体で 3 つ答えを返す

```
"aaaabaaabb".match(/.*?b\B/);
```

1	2	3	4	5	6	7	8	9	10
a	a	a	a	b	a	a	a	b	b

```
["aaaab"]
```

- 非貪欲なマッチング
- b の位置が単語境界以外のところ (\B) を探す
- 5 番目の b は単語境界にいないのでここまでがマッチ

```
"aaaabaaabb".match(/.*?b\B/g);
```

1	2	3	4	5	6	7	8	9	10
a	a	a	a	b	a	a	a	b	b

```
["aaaab", "aaab"]
```

- 繰り返しの探索をする
- b が単語境界にいないものを探す
- 10 番目の b は候補にならない。

```
"aaaabaaabb".match(/.*(?=b)/);
```

1	2	3	4	5	6	7	8	9	10
a	a	a	a	b	a	a	a	b	b

```
["aaaabaaab"]
```

- 任意の文字列 (`.*`) でそのあとが `b` であるもの (`(?=b)`) を探す
- 貪欲な探索なので 10 番目の `b` が `(?=b)` で指定されたもの
- マッチした部分にはこの部分が含まれない

```
"aaaabaaabb".match(/.*?(?=b)/);
```

1	2	3	4	5	6	7	8	9	10
a	a	a	a	b	a	a	a	b	b

```
["aaaa"]
```

- 非貪欲なマッチ
- (?=b) で指定された部分が 5 番目の b

```
"aaaabaaabb".match(/.*?(?=b)/g);
```

1	2	3	4	5	6	7	8	9	10
a	a	a	a	b	a	a	a	b	b

```
["aaaa", "", "aaa", "", ""]
```

- 非貪欲なマッチで繰り返しを行うもの
- 戻り値に空文字列があるのは、一度 (?=b) でマッチした処理を行った後、もう一度 b のところから探索を始めているためではないかと考えられる

```
"abccbcckkccaaMMaacc".match(/((.)\2).*\1/);
```

```
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
 a b c c b c c k k c  c  a  a  M  M  a  a  c  c
```

```
["ccbcckkccaaMMaacc", "cc", "c"]
```

- 正規表現の (.) は左から数えて 2 番目の括弧になる
- この部分にマッチした文字は \2 で参照できる
- (.)\2 は同じ文字が 2 つ並ぶものにマッチ
- この部分全体が再び括弧でくくられているので、この部分が \1 で参照できる
- この正規表現は同じ 2 つの文字で挟まれた文字列にマッチする
- 中央部の正規表現は任意の文字列を表す
- 貪欲なマッチなのではじめに現れる同じ文字が 2 つ続く 3,4 番目の cc が一番最後に現れる 18 番目と 19 番目の cc と組み合わされる
- グローバルな検索ではないときには戻り値に \1 と \2 が含まれる。

```
"abccbcckkccaaMMaacc".match(/((.)\2).*\1/g);
```

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 a  b  c  c  b  c  c  k  k  c  c  a  a  M  M  a  a  c  c
```

```
["ccbcckkccaaMMaacc"]
```

- g フラグが付いているが、条件に合うものは一つしかない
- 戻り値に \1 と \2 が含まれない

```
"abccbcckkccaaMMaacc".match(/((.)\2).*?\1/);
```

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 a  b  c  c  b  c  c  k  k  c  c  a  a  M  M  a  a  c  c
```

```
["ccbcc", "cc", "c"]
```

- 前問と異なり、中央部の任意の文字列が非貪欲になっている
- 3，4 番目と 6，7 番目の cc が対応
- 戻り値の配列の 2 番目と 3 番目は \1 と \2

```
"abccbcckkccaaMMaacc".match(/((.)\2).*?\1/g);
```

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 a  b  c  c  b  c  c  k  k  c  c  a  a  M  M  a  a  c  c
```

```
["ccbcc", "ccaaMMaacc"]
```

g が付いているのでさらに 10 番目と 11 番目、18 番目と 19 番目の cc が対応

```
"abccbcckkccaaMMaa".match(/((.)\2).*\1/);
```

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
 a  b  c  c  b  c  c  k  k  c  c  a  a  M  M  a  a
```

```
["ccbcckkcc", "cc", "c"]
```

- これまでの文字列から最後の 2 文字を取り除いた文字列で同じことを行っている
- 3,4 番目の cc に対応するのは 11，12 番目のもの

。

```
"abccbcckkccaaMMaa".match(/((.)\2).*\1/g);
```

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
 a  b  c  c  b  c  c  k  k  c  c  a  a  M  M  a  a
```

```
["ccbcckkcc", "aaMMaa"]
```

- g フラグが付いているのでマッチした部分列が返される
- 3,4 番目のと 10,11 番目の cc が対応
- そのあとの部分列で 12,13 番目と 16,17 番目の aa が対応する
- マッチした部分列は 2 つ

```
"abccbcckkccaaMMccaa".match(/((.)\2).*\1/g);
```

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 a  b  c  c  b  c  c  k  k  c  c  a  a  M  M  c  c  a  a
```

```
["ccbcckkccaaMMcc"]
```

- これまでの文字列と最後の 4 つが入れ替わっている
- 3,4 番目と 16,17 番目の cc が対応

```
"abccbcckkccaaMMccaa".match(/((.)\2).*?\1/g);
```

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 a  b  c  c  b  c  c  k  k  c  c  a  a  M  M  c  c  a  a
```

```
["ccbcc", "ccaaMMcc"]
```

- 前問と異なり、非貪欲な任意の部分文字列を途中に取る
- 3，4 番目と 6，7 番目の cc、10,11 番目と 16,17 番目の cc が対応する

エラー処理の例 (改良版)

2 回前の授業で次のエラー処理を紹介した。

```
function Person(name, y, m, d){
  if(name === "") throw new Error("名前がありません");
  this.name = name;
  this.year = y;
  if(m<1 || m>12) throw new Error("月が不正です");
  var date = new Date(y,m,0);
  if(d<1 || d>date.getDate()) throw new Error("日が不正
です");
  this.month = m,
  this.day = d
}
Person.prototype = {
...
}
```

エラー処理が不十分

このリストでは十分なエラー処理がなされていない。

- 数に変換されない文字列の入力に対しては値が `NaN` になる。
- その結果、たとえば月の値の評価 `m<1` は `false` となり、エラーチェックを通り抜けてしまう。
- 対処法としては `!(m>=1 && m<=12)` とすることも考えられるが、これでも小数点付きの数が排除できない。
- Web アプリケーションではテキストボックスの入力は文字列になるので、文字列の段階でチェックするほうが楽

Person2 の修正

Person オブジェクトの範囲を外部から入力させるときに文字列を数に変換する前に数値リテラルになっているかを判定することでプログラムが不正な値を受け付けないようにできる。
整数値だけにするのであれば正規表現は/^\d+$/である。

Person2 の修正

```
<title>エラーオブジェクト (改良)</title>
function Person(name, y, m, d){
  if(name === "") throw new Error("名前がありません");
  this.name = name;
  this.year = checkNum(y,1900,2020,"年");
  this.month = checkNum(m,1,12,"月");
  var date = new Date(y,this.month,0);
  this.day = checkNum(d,1,date.getDate(),"日");
}
function checkNum(s, low, high, mes) {
  if(s.match(/^\d+$/)) {
    if(s>=low &&s<=high) return s-0;//文字列を数字に変
換
  }
  throw new Error(mes+"が不正です");
}


...
```

解説

prompt() で戻ってきた文字列が数字だけからなっているかをチェックしたうえで、与えられた範囲内にあるかを調べている。

外部入力のチェックを！

- 外部からのデータの入力に対しては、データを吟味してから利用する
- 特に Web ページのテキストボックスからのデータ入力を利用して不正行為を行う手法が知られている