# ブログアプリを作ってみよう

## 1．サンプルデータをつくる

コマンドプロンプトで作業をおこないますので、まず、「環境変数」に「SQLite3」の実行ファイルがある場所を教えます。
「スタートメニュー」－「Windows システムメニュー」－「コントロールパネル」を開きます。
次に、「システム」－「システムの詳細設定」－「環境変数」をクリックします。
「Path」に「追加」として、以下の内容を追加します。

```
C:¥xampp¥MercuryMail
```

次に、アプリを作る場所を用意します。
「c:¥xampp¥htdocs」に「myblog」というフォルダを作成してください。
次に、エクスプローラーでそのフォルダを開き、URL 欄で、「cmd」と入力し、エンターキーを押してください。これでコマンドプロンプトが起動して、カレンドフォルダが「myblog」になっています。

次に、コマンドプロンプトで、（<Enter>は、キーボードの Enter キーを押すことです）

```
> sqlite3 -version <Enter>
```

と入力して、SQLite3 のバージョンが表示されたら、ＯＫです。
次に、Windows のコマンドプロンプトは文字コードが Shift-Jis なので、それを「UTF-8」変更します。
コマンドプロンプトを起動して、以下のように入力してください。

```
> chcp 65001 <Enter>
```

これで、文字コードが「UTF-8」なっているはずです。
それから、以下の作業をおこないます。
まず、SQLite3 を使って元となるサンプルデータベースを作ります。データベース名を「blog.db」とします。
コマンドプロンプトから、以下のようにコマンドを入力します。

```
> sqlite3 blog.db （Enter キー 以下、<Enter>）
```

つづいて、テーブルを作ります。各行の最後は<Enter>です。

```
sqlite> create table blog (
   ...> id integer primary key ,
   ...> title text ,
   ...> body text ,
   ...> date text ) ;
```

データを入力します。カラムを指定せずに、全項目を入力します。

```
sqlite> insert into blog values(1, '初めての投稿',
   ...>  'これが初めての投稿。これから初めてのブログアプリを作っていくよ。',
   ...> '2018-03-23 06:48' ) ;
```

今度は、カラム名を指定して入力してみます。
カラム id は、自動で入力するように設定してあります。（integer primary key）

```
sqlite> insert into blog (title, body, date) values(
   ...>  '2回めの投稿',
   ...>  'idを設定せずににデータを入力しているよ。うまくいくかな。',
   ...>  '2018-03-23 06:54' ) ;
```

データの一覧を見てみましょう。

```
sqlite> .header on
sqlite> .mode column
sqlite> select * from blog;
```

```
id title              body                                          date
-- ----------------- ---------------------------------------------- ----------------
1 初めての投稿    これが初めての投稿。これから初めてのブログア（略）   2018-03-23 06:48
2  2回めの投稿    idを設定せずににデータを入力しているよ。うま（略）   2018-03-23 06:54
```

うまくいきました。

## 2．データの一覧を表示するプログラムをつくる

それでは、今入力したデータを一覧するプログラムを作成してみます。
「manageBlog.php」というファイル名にします。

```
<?php
// manageBlog.php


$db = new PDO( 'sqlite:blog.db' );           // blog.dbに接続
$query = "select * from blog";           // テーブルblogからすべてのデータを読みだすクエリ文
$stmt = $db->query($query);           // クエリ文を実行。$resultに読み込む。
$blog = $stmt->fetch(PDO::FETCH_ASSOC);
var_dump($blog);
```

これで、ブラウザから、「http://localhost/myblog」とアドレスを指定すると、以下のような出力が出ます。

```
array(4) { ["id"]=> string(1) "1" ["title"]=> string(18) "初めての投稿" ["body"]=>
string(96) "これが初めての投稿。これから初めてのブログアプリを作っていくよ。" ["date"]=>
string(16) "2018-06-01 16:20" }
```

これをコピーし、テラパッドなどのエディタに貼り付けて、見やすく修正したのが、以下です。

```
array(4) {
  ["id"]=> string(1) "1"
  ["title"]=> string(18) "初めての投稿"
  ["body"]=> string(96) "これが初めての投稿。これから初めてのブログアプリを作っていくよ。"
  ["date"]=> string(16) "2018-06-01 16:20"
}
```

先ほど入力したデータの１件分であることが、わかります。
$blog = $stmt->fetch(PDO::FETCH_ASSOC) によって、１件分のデータを取り出すことができています。
その１件分のデータが存在する限り、繰り返して取り出すというふうにしたのが、以下です。

```
$db = new PDO( ‘sqlite:blog.db’ );            // blog.db に接続
$query = “select * from blog”;            // テーブル blog からすべてのデータを読みだすクエリ文
$stmt = $db->query($query);            // クエリ文を実行。$result に読み込む。
while ($blog = $stmt->fetch(PDO::FETCH_ASSOC)) {
     var_dump($blog);
}
```

これを先ほどと同じようにブラウザで表示し、その画面出力をコピーし、テラバッドに貼り付けて、見やすくしたのが、以下です。

```
array(4) {
    ["id"]=> string(1) "1"
    ["title"]=> string(18) "初めての投稿"
    ["body"]=> string(96) "これが初めての投稿。これから初めてのブログアプリを作っていくよ。
"
    ["date"]=> string(16) "2018-06-01 16:20"
}
array(4) {
    ["id"]=> string(1) "2"
    ["title"]=> string(16) "2 回めの投稿"
    ["body"]=> string(80) "id を設定せずにデータを入力しているよ。うまくいくかな。"
    ["date"]=> string(16) "2018-06-01 16:25"
}
```

これらはそれぞれ連想配列なので、以下のようにすると、うまく表示できます。

```
while ($blog = $stmt->fetch(PDO::FETCH_ASSOC)) {
     echo $blog[ ‘id’ ], “<br>”;
     echo $blog[ ‘title’ ], “<br>”;
     echo $blog[ ‘body’ ], “<br>”;
     echo $blog[ ‘date’ ], “<br>”;

}
```

このように表示されます。

```
1
初めての投稿
これが初めての投稿。これから初めてのブログアプリを作っていくよ。
2018-06-01 16:20
2
2 回めの投稿
id を設定せずにデータを入力しているよ。うまくいくかな。
2018-06-01 16:25
```

あとは、これを HTML とうまく組み合わせるだけです。

```
<?php
// manageBlog.php
```

```
$db = new PDO( 'sqlite:blog.db' );
$query =  "select * from blog" ;
$stmt = $db->query($query);
?>
<!doctype html>
<html lang=" ja" >
<head>
  <meta charset=" utf=8" >
  <title>MyBlog</title>
  <link rel=" stylesheet"  href=" myblog.css" >
</head>
<body>
  <div id=" wrap" >
    <header>
      <h1>MyBlog</h1>
    </header>
    <article>
      <?php
      // $stmt を実行して1レコードずつ連想配列に読み込む
      while ($blog = $stmt->fetch(PDO::FETCH_ASSOC)) {
      ?>
        <section>
          <div class=" id" >id:<?php echo $blog[ 'id' ]; ?></div>
          <h1 class=" title" ><?php echo $blog[ 'title' ]; ?></h1>
          <div class=" body" ><?php echo $blog[ 'body' ]; ?></div>
          <div class=" date" >作成: <time><?php echo $blog[ 'date' ]; ?></time></div>
        </section>
      <?php
      }
      ?>
    </article>
    <footer>
      <small>&copy; 2018 Seiichi Nukayama</small>
    </footer>
  </div><!-- #wrap -->
</body>
</html>
<?php
$db = null;          // データベースとの接続を解除する。
?>
```


こんな感じになっていると思います。

## 3．共通処理を別ファイルにして読み込むようにする

mylib.php というファイルを作って、そこに共通処理を記述して、各ファイルで読み込むようにします。

どういうことかというと、データベース名とかテーブル名は、固有名詞で、これから作るプログラムコードのいろんな場所で使います。間違って入力すると大変だし、このプログラムコードを他のときに使えるようにするためにも、別ファイルにして管理したほうがいいのです。

そこで、mylib.php というファイルを作ります。ほかの php ファイルと同じように作成してください。

そこに以下のようにデータベース名とテーブル名を登録しておきます。

mylib.php

```
<?php
$dbname = 'sqlite:blog.db' ;
$tablename = 'blog' ;
```

manageBlog.php は、以下のように修正します。

manageBlog.php

```
<?php
require_once('mylib.php');

$db = new PDO($dbname);
$query = "select * from $tablename) ;
$stmt = $db->query($query) ;
```

## 4．データベースの項目（カラム）を追加する

データベースに「カテゴリ」と「タグ」という項目を追加したいと思います。
カテゴリは大分類で、自分のブログ記事を大まかに分類するために使います。
タグは、小さな見出しで、ひとつのブログ記事にいくつでもタグをつけることができるものとします。
ただし、その場合は、半角空白で区切るものとします。

```
sqlite> alter table blog add column category text;
sqlite> alter table blog add column tag text;
```

新しく追加したカラムに、データを入力します。

この場合は既存のレコードにデータを修正する方法でやります。
idが1のレコードのカテゴリに「ブログ」を、タグに「新規作成」をセットします。
idが2のレコードのカテゴリに「ブログ」を、タグに「データの追加」をセットします。

```
sqlite> update blog set category = ‘ブログ’ where id = 1;
sqlite> update blog set tag = ‘新規作成’ where id = 1;
sqlite> update blog set category = ‘ブログ’ where id = 2;
sqlite> update blog set tag = ‘データの追加’ where id = 2;
```

sqlite> select * from blog; で、データを確認してみてください。
manageBlog.php も修正します。

```
  <section class="manageBlog">
    <div class="id">id: <?php echo $blog[ ‘id’ ]; ?></div>
    <h1 class="title"><?php echo $blog['title']; ?></h1>
    <div class="body"><?php echo $blog['body']; ?></div>
    <div class="date">作成: <time><?php echo $blog['date']; ?></time></div>
    <div class="category">カテゴリ：<?php echo $blog['category']; ?></div>
    <div class="tag">タグ：<?php echo $blog['tag']; ?></div>
  </section>
```

## 5．<header>と<footer>を共通化する

<header>部分と<footer>部分は、これからつくるいろいろなページで共通に使う部分です。そのたびに同じことを記述するのは無駄も多く、修正のときにも不便です。そこで、<header>部分と<footer>部分を外部ファイルにして、それを読み込むようにします。
以下の部分を manageBlog.php から切り取り、「header.php」とします。

header.php

```
<?php // header.php ?>
<!doctype html
<html lang="ja">
    <head>
        <meta charset="utf-8">
        <title>MyBlog</title>
        <link rel="stylesheet" href="myblog.css">
    </head>
```

```
    <body>
        <div id="wrap">
            <header>
                <h1>MyBlog</h1>
            </header>
            <article>
```

<article>の部分も header.php に含めるかどうかは悩むところですが、今回は、これでやってみたいと思います。
次に<footer>部分を「footer.php」に切り出します。

**footer.php**

```
<?php // footer.php  ?>
            </article>
            <footer>
                <small>&copy; 2018 Seiichi Nukayama</small>
            </footer>
        </div><!-- #wrap -->
    </body>
</html>
```

もとの manageBlog.php のそれぞれの部分には、以下のように、header.php、 footer.php を読み込むように記述を変更しておきます。

**header.php に記述した部分**

```
...（略）...
$stmt = $db->query($query);
<?php require_once  'header.php' ; ?>
```

**footer.php に記述した部分**

```
...（略）...
<?php require_once  'footer.php' ; ?>
<?php
$db = null;
```

## 6．myblog.css でデザインを記述する

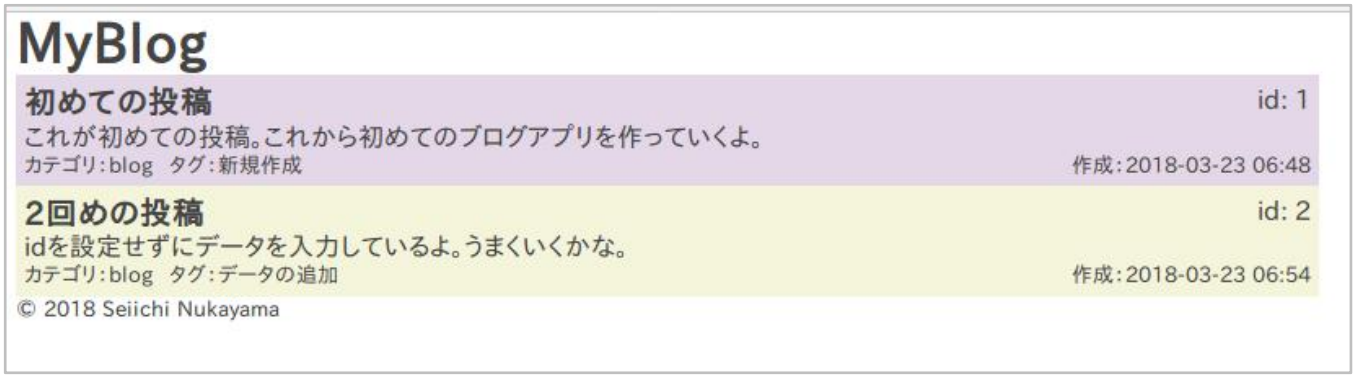
一覧を見るページ（manageBlog.php）のデザインをある程度考えておきます。myblog.css を新規作成して、以下のようにします。

**myblog.css**

```
@charset  "utf-8" ;

/* ===== 共通設定 ===== */
* {                        /* すべての要素のマージンとパディングをゼロにする */
  margin: 0;
  padding: 0;
}
```

```
ul {                        /* リストの黒丸をなしにする */
  list-style-type: none;
}
a {                         /* リンクの下線をなしにする */
  text-decoration: none;
}
img {                       /* 画像の下に出る余白をなしにする */
  vertical-align: bottom;
}
.clearfix:after {      /* <section>に 「class=" clearfix" 」 を追加してください。*/
  content: "" ;
  display: block;
  clear: both;
}
body {
  font-family: 'メイリオ', 'Hiragino Kaku Gothic Pro', sans-serif;
  color: #444;              /* 文字は真っ黒よりも、少し薄いほうがいい?  */
}
#wrap {
  width: 800px;             /*  body の中にボックスをつくり、幅を 800px にして中央寄せ */
  margin: 0 auto;
}

/* ===== manageBlog.php ===== */
.manageBlog .id {    /* <section>に「class=" clearfix manageBlog" 」を追加。 */
  float: right;
}
.manageBlog .date {
  float: right;
  font-size: 0.8em;
}
.manageBlog .category,
.manageBlog .tag {
  float: left;
  font-size: 0.8em;
}
.manageBlog .category {
  margin-right: 10px;
}
.manageBlog:nth-child(odd) {      /* 奇数 */
  background-color: #e4d8e8;
}
```

```
.manageBlog:nth-child(even) {      /* 偶数 */
  background-color: beige;
}
.manageBlog {
  padding: 5px;
}
```




## 7．単独の記事表示ページをつくる

単独の記事表示ページをつくります。ファイル名は「showPage.php」とします。

記事一覧ページ（manageBlog.php）の各記事のタイトル部分をクリックすることで、その単独記事が表示されるようにします。それをどのように実現するかですが、今回はシンプルに実現してみます。

タイトルをクリックすると、その記事の「id」が「showPage.php?id=<id>」の<id>部分に表すことができれば、GET で受け取ることができます。これを使います。

$blog[‘id’]を何回も使うのが邪魔くさいので、以下のように$id をつくります。

manageBlog.php

```
while ($blog = $stmt->fetch(PDO::FETCH_ASSOC)) {
    $id = $blog[ ‘id’ ];
?>
    <section class=” clearfix manageBlog” >
        <div class=” id” >id:<?php echo $id; ?></div>
```

次に、manageBlog.php のタイトル部分にリンクを設定します。

manageBlog.php

```
<div class="clearfix">
   <div class="id">id: <?php echo $id; ?></div>
   <h1 class="title">
     <a href="showBlog.php?id=<?php echo $id; ?>">
         <?php echo h($blog['title']); ?></a></h1>
</div>
```

タイトル部分にリンクを設定したので、文字色が変わってしまいました。黒い色にします。そして、マウスが上に乗ると、少し色が薄くなるようにします。これはすべての<a>タグについて共通デザインにしたいので、以下の内容を「共通設定」の部分に追加します。

myblog.css

```
a {
    text-decoration: none;
```

```
    color: #444;
}
a:hover {
    color: #888;
}
```

さて、いよいよ showBlog.php をつくります。manageBlog.php から id 値が URL で渡されるので、それを GET で受け取ります。

$id = $_GET[‘id’];

それをデータベースから探し出すのは、以下のコマンドです。

SELECT * FROM blog WHERE id = <id>;

「:id」というラベル名で、SQLite3 の命令文に入れることのできる変数のようなものです。

showBlog.php

```
<?php // showBlog.php  ?>
<?php
require_once 'mylib.php';

if (!empty($_GET['id'])) {
   $id = (int)$_GET['id'];
   $db = new PDO($dbname);
   $query = "select * from $tablename where id = :id";
   $stmt = $db->prepare($query);
   $stmt->bindValue(':id', $id, PDO::PARAM_INT);
   $stmt->execute();
   $row = $stmt->fetch(PDO::FETCH_ASSOC);
   var_dump($row);
}
```

とりあえず、$row の中身を見てみます。以下のようになっています。

```
array(6) {
  ["id"]=> string(1) "1"
  ["title"]=> string(18) "初めての投稿"
  ["body"]=> string(96) "これが初めての投稿。これから初めてのブログアプリを作っていくよ。"
  ["date"]=> string(16) "2018-06-01 16:20"
  ["category"]=> string(9) "ブログ"
  ["tag"]=> string(12) "新規作成"
}
```

1 件分のデータを取り出すことができています。あとは、もしデータを取り込むことができたならばならばという条件文をつくってやって、あとの処理をつくります。

あとの処理は、$row の連想配列の中身を、$id ... $tag に取り込みます。そして、それを HTML 文の中で表示することとします。

showBlog.php

```
<?php // showBlog.php  ?>
<?php
require_once 'mylib.php';

if (!empty($_GET['id'])) {
   $id = (int)$_GET['id'];
   $db = new PDO($dbname);
   $query = "select * from $tablename where id = :id";
   $stmt = $db->prepare($query);
   $stmt->bindValue(':id', $id, PDO::PARAM_INT);
   $stmt->execute();
   if ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
        $id = $row['id'];
        $title = $row['title'];
        $body = $row['body'];
        $date = $row['date'];
        $category = $row['category'];
        $tag = $row['tag'];
   }
   $db = null;
}

require_once 'header.php';
?>
<div class="id">id:<?php echo $id; ?></div>
<h1 class="title"><?php echo $title; ?></h1>
<div class="body"><?php echo $body; ?></div>
<div class="date">作成：<time><?php echo $date; ?></time></div>
<div class="category">カテゴリ：<?php echo $category; ?></div>
<div class="tag">タグ：<?php echo $tag; ?></div>

<?php  require_once 'footer.php';  ?>
```

このように表示されます。


形を整えます。
この article の中の部分を「single-page」というクラスでくくり、デザインしようと思います。

```
<div class=" single-page  clearfix" >      <!-- id を float させるため -->
  <div class=" id" >id:<?php echo $id; ?></div>
  <h1 class="title"><?php echo h($title); ?></h1>
  <div class="body"><?php echo h($body); ?></div>
  <div class="date">作成: <time><?php echo h($date); ?></time></div>
  <div class="category">カテゴリ: <?php echo h($category); ?></div>
  <div class="tag">タグ: <?php echo h($tag); ?></div>
</div><!-- .single-page -->
```

myblog.css

```
/* ========= showBlog.php ========= */
.single-page {
   width: 610px;
}
.single-page h3 {
   font-size: 1em;
   font-weight: normal;
}
.single-page h1 {
   width: 600px;
   font-size: 1.2em;
   border-left: solid 10px #aaa;
   padding-left: 10px;
   margin-bottom: 5px;
}
.single-page .body {
   width: 600px;
   height: 400px;
   border: solid 1px #aaa;
   margin-bottom: 5px;
}
```

```
.single-page .date {
   float: right;
}
.single-page  .id {
      float: right;
}
```


## 8．データを追加する

新規データを入力するページを作ります。「inputBlog.php」とします。新規データはフォームで入力します。とりあえずは、以下のようになるかと思います。

inputBlog.php

```
<?php  // inputBlog.php
require_once 'mylib.php';                                    // mylib.php を読み込む

 require_once 'header.php';                                  //  ヘッダー部の読み込み
?>
<h1 class="inputBlog-h1">新規作成</h1>
<form action="" method="post">                               <!-- action 属性はこれから考える -->
    <label for="form-title">タイトル:</label><br>
    <input type="text" name="title" id="form-title" required><br>     <!-- 必須項目 -->

    <label for="form-body">内容:</label><br>
    <textarea name="body" id="form-body" required></textarea><br> <!-- 必須 -->

    <label for="form-category">カテゴリ:</label><br>
   <input type="text" name="category" id="form-category" required><br>

<!-- 必須 -->
```

```
    <label for="form-tag">タグ:</label><br>
    <input type="text" name="tag" id="form-tag" required><br>        <!-- 必須項目 -->
                                                                    <!-- 日時は date 関数で取得してテキストに -->
  作成: <input type="text" name="date" id="form-date"
            value="<?php echo date("Y-m-d H:i"); ?>"><br>

  <input type="submit" value="作成" id="form-submit">
  <a href="manageBlog.php" id="form-cancel">                        <!-- 入力作業の取消ボタン -->
    <button type="button">取消</button></a>
</form>

<?php require_once 'footer.php'; ?>                                 <!-- フッター部の読み込み -->
```


スタイルシートにデザインを記述します。

myblog.css

```
/* ========== inputBlog.php ============ */
.inputBlog-h1 {
    font-size: 1em;
}
.inputBlog-h1:before {                                  /* 飾り付け・・・他にいいデザインがあれば、 */
content: "---// ";                                                  /* それでもよい  */
}
.inputBlog-h1:after {
    content: " //---";
```

```
}
#form-title {
    width: 600px;
}
#form-body {
    width: 600px;
    height: 400px;
}
#form-category {
    width: 200px;
}
#form-tag {
    width: 400px;
}
#form-submit {
    width: 50px;
height: 50px;
    cursor: pointer;
    margin-right: 10px;
}
#form-cancel {
    display: inline-block;
}
#form-cancel button {
    width: 50px;
    height: 50px;
    cursor: pointer;
}
```


この入力画面からデータを登録する処理へとすすむわけですが、そのプログラムを「insertBlog.php」とします。したがって、inputBlog.php の<form>タグの action 属性を未指定のままにしていましたが、それを以下のようにしてください。

```
<form action="insertBlog.php" method="post">
```

さて、insertBlog.php の記述にとりかかります。

**insertBlog.php**

```
<?php // insertBlog.php
require_once 'mylib.php';

$okcount = 0;
// それぞれ POST データが空でなければ、変数にセット
if (!empty($_POST['title'])) {        $title = $_POST['title'];    $okcount++; }
if (!empty($_POST['body'])) {      $body = $_POST['body']; $okcount++; }
if (!empty($_POST['date'])) {       $date = $_POST['date'];  $okcount++; }
if (!empty($_POST['category'])) { $category = $_POST['category'];     $okcount++; }
if (!empty($_POST['tag'])) {          $tag = $_POST['tag'];    $okcount++; }

// okcount が 5 ということは、すべての変数がセットできたということ
if ($okcount === 5) {
   // データベースに接続
   $db = getDB();
   // prepare という方法でデータをセット。セキュリティと正確さのため。推奨。
   $query = "insert into " . TABLENAME . " (title, body, date, category, tag) values
(?, ?, ?, ?, ?)";
   $stmt = $db->prepare($query);
```

```
        // ?の順番にデータをセットできる
    $stmt->bindValue(1, $title, PARAM_STR);
    $stmt->bindValue(2, $body, PARAM_STR);
    $stmt->bindValue(3, $date, PARAM_STR);
    $stmt->bindValue(4, $category, PARAM_STR);
    $stmt->bindValue(5, $tag, PARAM_STR);
    $stmt->execute();
    $msg = "登録しました。";
    $db->close();
} else {
    $msg = "未入力の項目があったので、データベースには登録しませんでした。";
}
```

これで、データベースに登録できたのですが、今登録したデータを表示しなければなりません。これは、showBlog.php に id を渡して実現したいと思います。また、$msg も合わせて渡します。つまり、以下のような記述になります。

```
header( “Location: showBlog.php?id={$id}&msg={$msg}” );
```

header()というのは php の関数で、引数に‘Location: <URL>’を指定すると、その<URL>に処理がいくのです。こういうのを「リダイレクト」といいます。

となると、今登録したてのレコードの「id」がわかればいいことになります。

この中の id は、テーブル定義のときに「id integer primari key,」として、自動連番に設定しています。ですから、新規データを入力するときも、id は指定していません。となると、どのようにして今登録したてのレコードの id を知ることができるかですが、SQLite3 には、last_insert_rowid() という関数が用意されていて、今登録したての rowid を知ることができるのです。rowid というのは、SQLite3 が id とは別に自動的に用意している id のようなもので、ユーザーからは設定できません。で、以下のようなクエリ文で、取得できます。

sqlite> select id from テーブル名 where rowid = last_insert_rowid();

これを使って、さきほどの記述に以下の部分を付け加えます。

```
$msg = “登録しました。”
$query = “select id from “ . TABLENAME . “ where rowid = last_insert_rowid()”;
$stmt = $db->query($query);
if ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    $id = $row[ ‘id’ ];
}
$db = null;
} else {
    $msg = “未入力の項目があったので、データベースには登録しませんでした。”;
}
header( “Location: showBlog.php?id={$id}” );
```

こんな画面が出るかと思います。


ついでに、$msg も表示できるように、しておきます。

```
header( “Location: showBlog.php?id={$id}&msg={$msg}” );
```

$msg の表示は、ヘッダー部とアーティクル部の間に組み込むことにします。

```
...（略）...
</header>
<div class=” notice” ><?php if (!empty($_GET[ ‘msg’ ])) echo $_GET[ ‘msg’ ]; ?></div>
<article>
...（略）...
```

デザインは、とりあえず、以下のようにしておきます。

myblog.css（共通設定の部分に追加）

```
.notice {
    float: right;
    color: lightgreen;
}
```

## 9．ヘッダー部分にちょっと追加

新規作成のボタンをヘッダー部分につけてみます。

header.php

```
<header>
   <h1>MyBlog</h1>
   <div class="newBlog"><a href="inputBlog.php">[ NEW ]</a></div>
</header>
```

タイトルが英語なので、新規作成ボタンも英語にしてみました。

ついでに、タイトルを少し小さくして、その横に「NEW」を配置し、ヘッダーの下に余白をつけまし

た。

myblog.css

```
/* ========== ヘッダー ============ */
header {
    position: relative;
    margin-bottom: 10px;
}
header .newBlog {
    position: absolute;
    top: 6px;
    left: 140px;
}
header h1 {
    font-size: 1.5em;
}
```

これを mybloc.css の「共通設定」の下に付け加えてください。

また、$msg を表示する部分を作っておきます。以下のように、ヘッダーとアーティクルの間に入れておきます。

header.php

```
...（略）...
</header>
<div class=" notice" >
    <?php if (!empty($_GET[ 'msg' ])) echo $_GET[ 'msg' ]; ?>
</div>
<article>
```

## 10.更新画面（編集ページ）をつくる

単独表示ページができたので、編集ページもつくります。記事を訂正したくなったときに、この編集ページを使うことになります。

単独表示ページに編集ページへのリンクをつくります。このリンクは showBlog.php のタイトルの右端に表示させましょう。画像を用意しましたので、それを使ってください。

更新画面には GET ではなく、POST で値を渡すことにします。内容を書き換える処理なので、少しセキュリティに配慮するのです。（本来なら、ここで id とパスワードを要求するところです。これはあとで考えることにします）

POST のやり方ですが、<button>を使ってみます。<form>をつくり、action 属性で次の php を指定します。

showBlog.php

```
<div class="single-page clearfix">
  <div class="editThis">
      <form action=" inputBlog.php"  method=" post" >
          <button type=" submit"  name=" id"  value=" <?php echo $id; ?>" >
            <img src=" img/pencil.png"  alt=" EDIT" >
        </button>
```

```
        </form>
    </div>
    <div class="id">id:<?php echo $id; ?></div>
```

タイトルの横の方に配置します。

myblog.css

```
/* ===== showBlog.php ===== */
.single-page .id {
   float: right;
    display: none;          // idは表示しないでおきます。（特に必要ないでしょう）
}
.single-page .editThis {
    float: right;
}
.single-page .editThis button {
   background: none;      // ボタンの背景をなしにする
   border: none;               //  ボタンの枠線をなしにする
   cursor: pointer;            // ボタンの上にマウスがきたら、手の形にする
}
.single-page .editThis button:hover img {
   opacity: 0.7;             // ボタンの上にきたら、画像を薄くする
   width: 120%;           // 画像を大きくする
}
```


さて、編集のためのプログラムを作成します。ファイル名は「editBlog.php」としましょう。inputBlog.php をコピーして作ります。<form>のところが同じだからです。

showBlog.php から id を POST で受け取ります。その id 値により、データベースを検索して、データを取得します。この部分は、showBlog.php とほとんど同じです。showBlog.php の部分をコピーして貼り付けます。

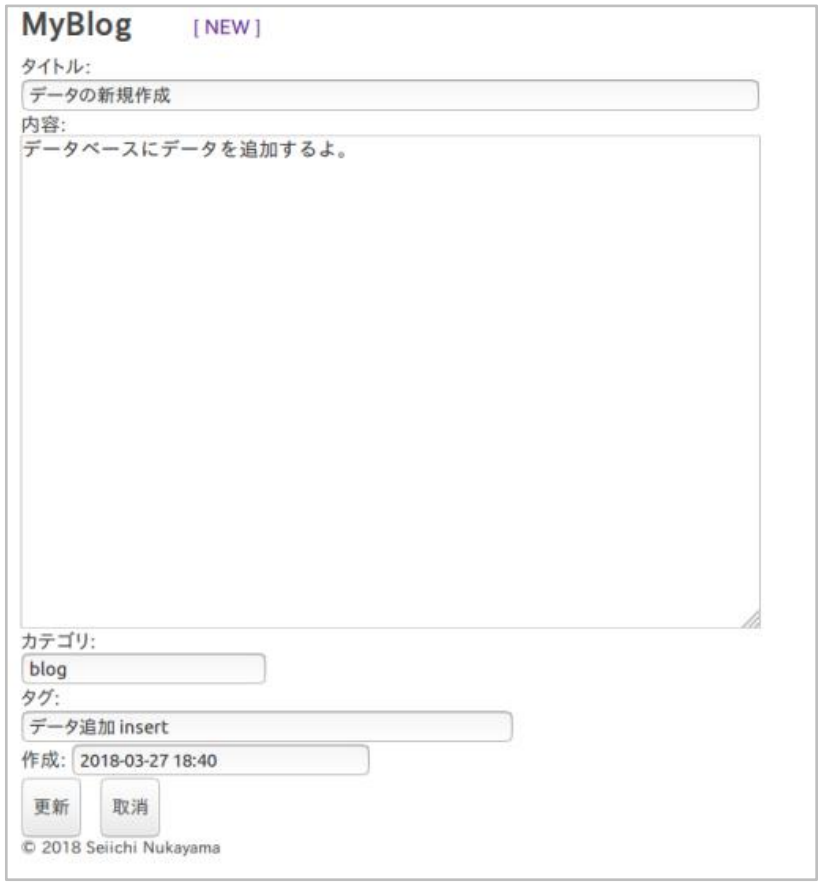
<form>の action 属性は、updateBlog.php とします。また、画面タイトルは「編集」にします。submit も「更新」とします。さらに、id 値を post データとしてわたすので、それは hidden 属性でわたすことにします。

editBlog.php

```
<?php  // editBlog.php
require_once('mylib.php');

if (!empty($_POST['id'])) {
    $id = (int)$_POST['id'];
   $db = new PDO($dbname);
   $query = "select * from $tablename where id = :id";
   $stmt = $db->prepare($query);
   $stmt->bindValue(':id', $id, PDO::PARAM_INT);
   $stmt->execute();
   if ($row = $stmt->fetch(PDO::FETCH_ASSOC) {
        $id = $row['id'];
        $title = $row['title'];
        $body = $row['body'];
        $date = $row['date'];
        $category = $row['category'];
        $tag = $row['tag'];
   }
   $db = null;
}
<h1 class=" inputBlog-h1" >編集</h1>
<form action=" updateBlog.php"  method=" post" >
   <input type=" hidden"  name=" id"  value=" <?php echo $id; ?>" >

    <label for="form-title">タイトル:</label><br>
    <input type="text" name="title" id="form-title" required value="<?php echo
$title; ?>"><br>

    <label for="form-body">内容:</label><br>
    <textarea name="body" id="form-body" required><?php echo $body; ?></textarea><br>

    <label for="formcategory">カテゴリ:</label><br>
    <input type="text" name="category" id="form-category" required value="<?php echo
$category; ?>"><br>

    <label for="form-tag">タグ:</label><br>
```

```
    <input type="text" name="tag" id="form-tag" required value="<?php echo
$tag; ?>"><br>

    作成:<input type="text" name="date" id="form-date"
                value="<?php echo date("Y-m-d H:i"); ?>"><br>

    <input type="submit" value="更新" id="form-submit">
    <a href="manageBlog.php" id="form-cancel">
        <button type="button">取消</button></a>
</form>
.....（略）......
```

編集画面です。




次は、updateBlog.php です。これは、insertBlog.php とほぼ同じですが、一部分ちがうところがあります。それは、id 値が決まっているところです。insertBlog.php では、新しい id 値になるので、それは SQLite3 が自動的に id 値を作成してくれています。今回は、id 値を指定して更新しているので、editBlog.php からは、input 文の hidden 属性で id 値がわたされます。それを$_POST で受け取らなければなりません。insertBlog.php をコピーして、updateBlog.php を作り、修正することにしましょう。

また、表示の部分は、わざわざ作らなくても showBlog.php にリダイレクトすればいけます。
この時、inputBlog.php では、id 値を rowid()で調べなければなりませんでしたが、今回は、id 値はわかっているので、その部分の記述は削除できます。（以下の記述は不要なので削除）

```
$query = “select id from “ . TABLENAME . “ where rowid = last_insert_rowid()”;
$stmt = $db->query($query);
if ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    $id = $row[ ‘id’ ];
}
```

updateBlog.php

```
<?php // updateBlog.php  ?>
<?php
require_once('mylib.php');
$okcount = 0;
if (!empty($_POST['id'])) { $id = (int)$_POST['id']; $okcount++; }
if (!empty($_POST['title'])) { $title = $_POST['title']; $okcount++; }
```

```
if (!empty($_POST['body'])) { $body = $_POST['body']; $okcount++; }
if (!empty($_POST['date'])) { $date = $_POST['date']; $okcount++; }
if (!empty($_POST['category'])) { $category = $_POST['category']; $okcount++; }
if (!empty($_POST['tag'])) {   $tag = $_POST['tag']; $okcount++; }
if ($okcount === 6) {
   $db = new PDO($dbname);
   $query = "update $tablename set title = :title, body = :body,  "
                 .  "date = :date, category = :category, tag = :tag where id = :id";
   $stmt = $db->prepare($query);
   $stmt->bindValue(':title', $title, PDO::PARAM_STR);
   $stmt->bindValue(':body', $body, PDO::PARAM_STR);
   $stmt->bindValue(':date', $date, PDO::PARAM_STR);
   $stmt->bindValue(':category', $category, PDO::PARAM_STR);
   $stmt->bindValue(':tag', $tag, PDO::PARAM_STR);
   $stmt->bindValue(':id', $id, PDO::PARAM_INT);
   $stmt->execute();
   $msg = "更新しました。";
   $db = null;
} else {
   $msg = "未入力の項目があったので、更新しませんでした。";
}
header( "Location: showBlog.php?id=[$id]&msg={$msg}" );        // showBlog.php へリダイレクト
?>
```

## 11.一覧ページ（manageBlog.php）をちょっと修正

毎回 URL 欄に入力するのも大変なので、タイトル部分をクリックすれば、manageBlog.php を呼び出せるようにしておきましょう。（リンクのデザインは記述済みです）

header.php

```
<h1><a href=" manageBlog.php" >MyBlog</a></h1>
```

また、一覧ページに本文（body 部）を表示すると冗長になり、大変だし、不必要だと思うので、削除します。

manageBlog.php

```
</div>
  <div class="body"><?php echo h($blog['body']); ?></div>  <-- 削除します。
  <div class="date">作成: <time><?php echo h($blog['date']); ?></time></div>
```

また、「id」の表示も必要ないでしょう。これは、display: none; で消しておきます。（こうしておくと、もし必要なとき、display: block; で再度表示できます）

myblog.css

```
.manageBlog .id {
    float: right;
    display: none;      <== 表示を消しておく。
}
```

## 12.検索ボックスをつくる

記事がたくさんになってくると、自分がどこに書いたかわからなくなります。そのために検索機能をつくっておきます。検索対象は、タイトル・本文・カテゴリ・タグとします。このプログラムコードを別ファイル findBlog.php として作り、keader.php に読み込むようにします。

findBlog.php

```
<?php // findBlog.php ?>
<div class="findBlog">
  <form action="manageBlog.php" method="post" class="clearfix">
      <div class="findword">
          <select name="findOf" id="findOf">
               <option value="title">タイトル</option>
               <option value=" body" >本文</option>
               <option value="category">カテゴリ</option>
               <option value="tag">タグ</option>
           </select>
       </div>
       <div class="findword">
           <input type="text" name="word">
       </div>
       <div class="findword">
           <input type="submit" value="検索">
       </div>
```

```
    </form>
</div>
```

これをヘッダー部の中にいれます。

header.php

```
 <header>
    <h1><a href="manageBlog.php">MyBlog</a></h1>
     <div class="newBlog"><a href="inputBlog.php">[ NEW ]</a></div>
    <?php require_once 'findBlog.php'; ?>
 </header>
```

myblog.css

```
/* ========== findBlog.php ============== */
.findword {
  float: left;
  margin-left: 2px;
}
.findBlog {
    position: absolute;
    top: 10px;
    right: 10px;
}
.findBlog input[type=" text" ] {
  height: 25px;
  width: 200px;
}
```

この検索ボックスのフォームは manageBlog.php で受け取ります。managaBlog.php に受け取るための記述を入れます。

manageBlog.php

```
$db = new PDO($dbname);
$query = "select * from $tablename";
$stmt = $db->query($query);

// findBlog から検索ワードを受け取ったら、$stmt を作りなおす
if (!empty($_POST['findOf']) && !empty($_POST['word'])) {
   $stmt = null;                                    // $stmt をいったん空にして. . .
    $findOf = $_POST['findOf'];     // タイトル・本文・カテゴリ・タグ
    $word = '%' . $_POST['word'] . '%';    // キーワードを含む文字列を検索する

    $query = "select * from $tablename where $findOf like :word";
    $stmt = $db->prepare($query);
    $stmt->bindValue(':word', $word, PDO::PARAM_STR);
```

```
    $stmt->execute();                    // 検索を実行
}
```

これで検索機能が使えるようになったかと思います。


## 13.削除ボタンをつくる

ブログ記事を削除できるようにします。
削除ボタンはアイコンで表示しましょう。イラストレーターでゴミ箱のアイコンをつくります。このアイコンは、manageBlog.php の各タイトルの右のところに表示するようにします。今は非表示にしていますが、ここには「id」が表示されていました。ここにゴミ箱のアイコンを表示します。ここならサイズは 20px 平方になります。このアイコンを「trash.png」とします。
アイコンができたら、この trash.png を img フォルダに入れて、表示させるようにします。

manageBlog.php

```
<section class="manageBlog clearfix">
      <div class="id">id: <?php echo $id; ?></div>  <!-- display: none; で非表示 -->
      <div class="trash">
            <form action="deleteBlog.php" method="post">
                <button type="submit" name="id" value="<?php echo $id; ?>">
                    <img src="img/trash.png" alt="削除"></button>
            </form>
      </div>
      <h1 class="title">
```

myblog.css

```
.manageBlog .id {
    float: right;
    display: none;          /* idは非表示にしている */
}
.manageBlog .trash {
    float: right;
}
.manageBlog .trash button {
```

```
    background: inherit;
    border: none;
    cursor: pointer;
}
```

manageBlog.php

showBlog.php へのリンクには「?id=<?php echo id; ?>」という文字列をつけて、GET で id を取得するようにしました。今回の削除プログラム「deleteBlog.php」にも同じやり方が考えられるのですが、削除するブログ記事の id を URL で指定できるのは、危険でしょうね。悪意のある第 3 者が適当な id で url に入力し、勝手に記事を削除するかもしれないというのはもちろん、自分でも番号をまちがって削除してしまうかもしれません。

そこで、post による受け渡しで id を指定するようにします。

<form action="deleteBlog.php" method="post">で、フォームを作成し、action 属性で処理をわたすファイル名を指定します。

<form>の中に、<button>をつくります。type="submit"とすると、送信ボタンと同じ機能を持たせることができます。name 属性で deleteBlog.php に渡すためのコントロール名を指定します。今回は「id」としました。そして、deleteBlog.php に渡す値を<?php echo $id; ?>で表します。

<button>には、画像を指定することもできます。trash.png を<img>タグで指定します。

myblog.css

<button>要素には、デフォルトでボタンとしてのデザインが適用されています。今回は画像を表示しますので、そのデザインが邪魔になるので、なくします。

background: inherit; -- 背景の設定を親要素と同じにする。このことで、ボタンに設定されていたグラデーションや背景色が解除されます。

border: none; -- ボタンについていた枠線をなくします。

cursor: pointer; -- ゴミ箱の絵の上にマウスをのせると、指の形になるようにします。

削除のためのプログラムをつくります。

$_POST['id']が空ではなかったら、削除処理をおこないます。(1)

データベースに接続します。(2)

削除の SQL 文は、「DELETE FROM テーブル名 WHERE id = 番号」です。(3)(4)

これを今回も prepare 文でおこないます。(4)

deleteBlog.php

```
<?php // deleteBlog.php
require_once 'mylib.php' ;

if (!empty($_POST['id'])) {
    $id = (int)$_POST['id'];                                                    -----
(1)
    $db                    =                    new                    PDO($dbname);
----- (2)
    $query = "delete from $tablename where id = :id";  ----- (3) (4)
    $stmt = $db->prepare($query);                                          ----- (4)
    $stmt->bindValue(':id', $id, PDO::PARAM_INT);                  ----- (4)
    $stmt->execute();                                                  ----- (4)
   $db = null;
   $msg = "削除しました。";
```

```
}
else {
   $msg = “削除できませんでした。”;
}

header(“Location: manageBlog.php?msg={$msg}”);                                    -
---- (5)
exit();
?>
```

削除が成功したら、manageBlog.php に戻ることにします。(5)
ある指定された URL に移動することを「リダイレクト」といいます。実現方法にはいろいろありますが、
今回は、php の header 関数でリダイレクトします。

```
header( ‘Location: <URL>’ );
exit();
```

<URL>の部分には、「http://.....」という指定をします。今回はファイル名だけの指定をします。また、$msg をわたすようにします。
リダイレクトしたあとは、exit(); でそのスクリプトの処理を終了します。

### ヘッダー部の微調整をします。

$msg を表示する部分が、うまくいかないので、ヘッダー部の中に入れてしまいます。header.php を修正します。 ...（略）... を<?php require_once ‘findBlog.php’; ?>の上に移動します。

header.php

```
...（略）...
    <div class=” notice” >
        <?php if (!empty($_GET[ ‘msg’ ])) echo $_GET[ ‘msg’ ]; ?>
    </div>
    <?php require_once  ‘findBlog.php’ ; ?>
</header>
<article>
```

次に、myblog.css の.notice の部分を修正します。
共通部分に記述していた .notice { ..... }をヘッダー部分に移動します。そして、以下のように修正してください。

myblog.css

```
header .notice {
    position: absolute;
    left: 300px;
    bottom: 10px;
    color: lightgreen;
}
```

これで、見やすくなったかと思います。
ついでに、ヘッダー部の高さも微調整しておいてください。

header { height: 40px; } このぐらいでいいと思います。

### 確認ダイアログを表示させる

これで削除処理はできましたが、ゴミ箱のアイコンをクリックしたときに、確認ダイアログがほしいところです。これは php ファイルを作って、確認用のページに移り、そこでまた確認ボタンを押して・・・というふうにもできますが、別のページが開くのが大げさすぎます。ここは、小さな確認ダイアログがいいですね。
実は Javascript には、フォームの送信ボタンを押したときの確認用スクリプトが用意されているのです。manageBlog.php のフォームのところに、以下の記述を付け加えます。

onSubmit="return kakunin()"

kakunin() というのは、これから記述する javascript の関数です。myfunc.js という名前のファイルをつくり、そこに以下のように avascript のスクリプトを記述します。

**myfunc.js**

```
// myfunc.js
// このファイルは、manageBlog.php 用である。
function kakunin() {
    if (window.confirm('削除します。よろしいですか? ')) {
        return true;
    }
    else {
        return false;
    }
}
```

この myfunc.js を読み込む設定をします。header.php に以下の記述を加えてください。

**header.php**

```
...(略)...
    <link rel=" stylesheet"  href=" myblog.css" >
    <script src=" myfunc.js" ></script>
</head>
...(略)...
```

次に、manageBlog.php の削除の<form>処理のところに、以下の記述を追加します。

**manageBlog.php**

```
...(略)...
<form action="deleteBlog.php" method="post" onSubmit="return kakunin()">
...(略)...
```

ゴミ箱ボタンをクリックすると、まず onSubmit で指定してある kakunin()関数が処理されます。
kakunin()関数では、confirm 関数が「削除します。よろしいですか? 」という文字を引数にとって起動します。confirm 関数では「はい」「いいえ」がデフォルトで表示され、「はい」をクリックすると「true」が戻り値となります。「いいえ」がクリックされると「false」が戻り値となります。
kakunin()関数から処理が onSubmit のところに戻り、「true」の値が返ってきた場合に<form>が処理されるというわけです。

## 14.Markdown 記法を使えるようにする

ブログの記事は今のところテキスト形式で表示するしかありません。しかし、文字を大きくしたり、見やすくするために、いろいろとデザインしたいところです。これをどのように実現するかですが、見出しの部分に<h1>～<h6>を設定したり、箇条書きを<ul><li>～</li></l>でマークアップしたり、HTML のタグでマークアップできれば、スタイルシートでデザインできます。
実際にやってみますと、タグがそのまま表示され、タグとしての働きをしていません。これは、htmlspecialchars 関数の働きによるもので、これにより悪意のあるスクリプトがはいっていたら無効化しているわけです。だから、HTML のタグも無効化しているわけです。
となると、htmlspecialchars 関数で処理したあとに、HTML のタグを付与すればいいわけですが、そのためには、<h1>なら、#、<h2>なら##というように、何らかの記号を考えて、その記号を文章につけておいて、それを画面に出力するときに<h1>などのタグに変換できればいいわけです。
実は、そういうことを考えた人がいて、そのうちのひとつに Markdown 記法というのがあり、現在広く使われています。今回はこれを利用したいと思います。
php で使える Markdown にもいろいろあるようですが、「PHP Markdown」(https://michelf.ca/projects/php-markdown/)を使います。ここから「PHP Markdown Lib 1.8.0」(4 月 2 日現在)をダウンロードし、解凍したら、その中の「Michelf」フォルダを現在作業しているフォルダにコピーします。
次に、表示を受け持っているのは、「showBlog.php」ですので、そこにこの php-Markdown を読み込みます。記述のしかたは php-Markdown のページに書かれてあります。

showBlog.php

```
require_once( 'mylib.php' );
require_once( 'Michelf/MarkdownExtra.inc.php' );
use Michelf¥MarkdownExtra;
```

これで php-Markdown が使えるようになってます。
この php-Markdown は、$body を読み込んで、それに htmlspecialchars 関数で処理したあとにこれで処理します。以下のようになります。

showBlog.php

```
$newbody = MarkdownExtra::defaultTransform(h($body));
require_once( 'header.php' );
```

$body はすでに htmlspecialchars 関数で処理しているので、showPage.php での処理はなくし、この $newbody を表示するように書き換えます。

showPage.php

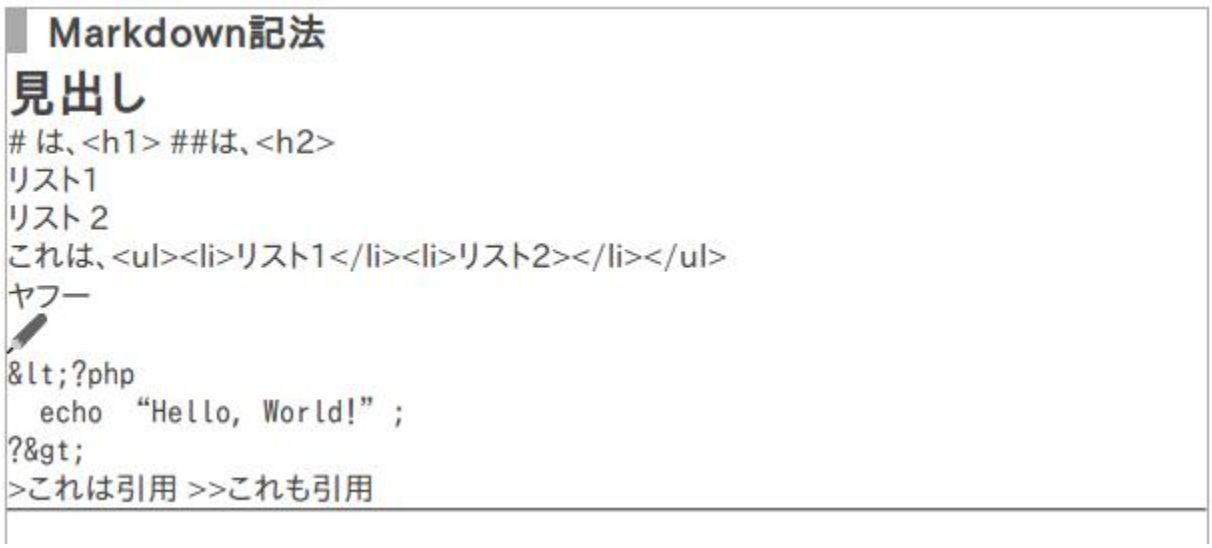
```
<div class-" body" ><?php echo $newbody; ?></div>
```

では、Markdown 記法で書いてみます。
[NEW]をクリックして新しい記事を書きましょう。タイトルは「Markdown 記法」とでもしましょう。
本文に以下のように記述してください。

```
# Markdown 記法
## 見出し
¥# は、<h1>    <-- 半角スペース２つ(改行)
¥##は、<h2>
```

```
- リスト 1
- リスト 2

これは、<ul><li>リスト 1</li><li>リスト 2></li></ul>

[ヤフー](http://www.yahoo.co.jp)

![鉛筆](img/pencil.png)

```
<?php
  echo “Hello, World!”;
?>
```

>これは引用
>>これも引用
***
```

カテゴリには、blog。タグには、Markdown。とでも入れときましょう。
これで登録すればどうなるでしょうか?




「ページのソースを表示」でみるとわかりますが、

```
<h1>Markdown 記法</h1>
<h2>見出し</h2>
<ul>
<li>リスト 1</li>
<li>リスト 2</li>
</ul>

<p><a href="http://www.yahoo.co.jp">ヤフー</a></p>
```

```
<p><img src="img/pencil.png" alt="鉛筆" /></p>
```

このようにHTMLのタグが出力されているのがわかります。あとは、myblog.cssのなかでデザインを指定してやれば表示を工夫することができます。
しかし、以下の部分は、HTMLのタグが出力されていません。

```
<pre><code>&amp;lt;?php
  echo “Hello, World!”;
?&amp;gt;
</code></pre>
<p>&gt;これは引用
&gt;&gt;これも引用</p>
```

これは、たとえば、「<?php」という部分があれば、これをhtmlspecialchars関数で「<?php」という文字列に変えて、それをさらにMarkdownExtraで「&」を「&」に変えたので、結局「<?php」というふうになっているのです。つまり、タグの無効化を二重でやってしまっているのです。ですから、ソースコードを出力する部分は、htmlspecialchars関数で処理したのを元に戻してやって、それからMarkdownExtraで処理すればいいわけです。
その働きをする関数をつくりましたので、それを「mylib.php」に加えてください。なお、引用を表す「>」もhtmlspecialchars関数の処理を戻すようにしておきました。
ファイル名「unEscMark.php」をエディタで開き、すべてを選択して、mylib.phpに貼り付けてください。
次に、showBlog.phpの$newbodyのところを以下のように変更します。

**showBlog.php**

```
$tmpbody = unEscMark(h($body));
$newbody = MarkdownExtra::defaultTransform($tmpbody);
```

これで、ソースコードの部分と引用の部分については、htmlspecialchars関数の処理を元に戻せました。
その部分のスタイルシートをmymarkdown.cssとして用意したので、それを読み込む記述をheader.phpに加えてください。

**header.php**

```
<link rel=” stylesheet”  href=” myblog.css” >
<link rel="stylesheet" href="mymarkdown.css">
```

これで、うまく表示できたはずです。

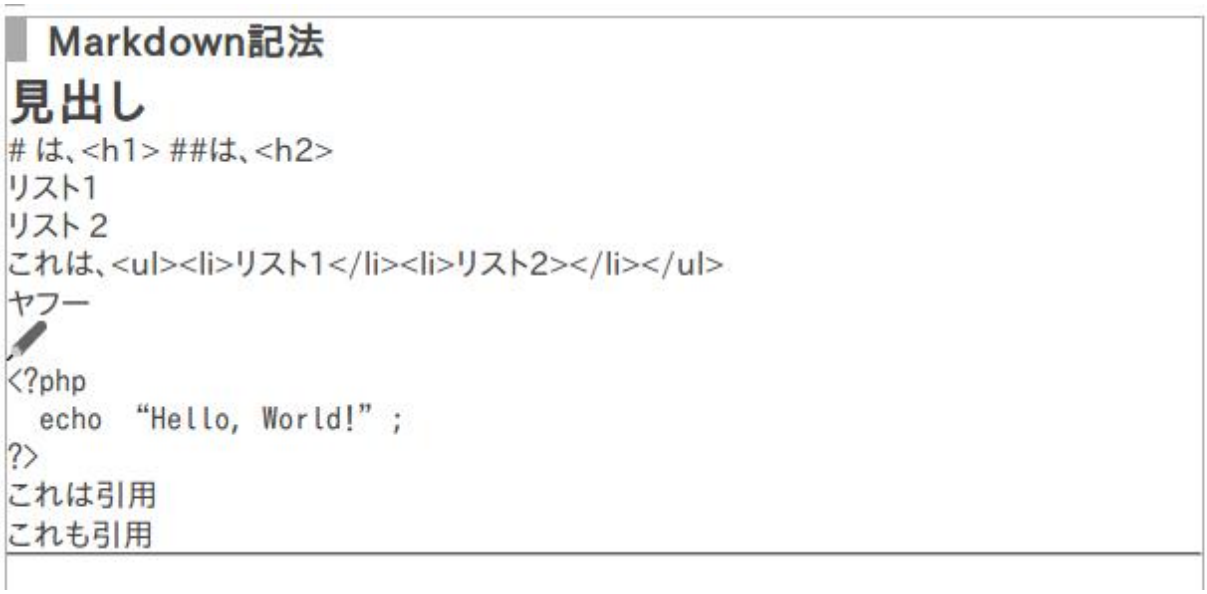
**注）**XAMPPのバージョンによって、unEscMark関数の最後のところが違ってきます。
最新バージョンのXAMPPの場合

```
return $newStr;
```

以前のバージョンのXAMPPの場合

```
return implode($newStr);
```

この原因はまだ調査中ｄす。




Markdown 記法について
Markdown 記法 日本語 Markdown ユーザー会　http://www.markdown.jp/syntax/
かんたん Markdown の記法　http://tatesuke.github.io/KanTanMarkdown/syntax.html

## 15.新規入力や更新についてはパスワード必須とする

showBlog.php は誰が見てもかまいませんが、inputBlog.php は誰でもアクセスできては大変です。また、deleteBlog.php も同様です。そこで、これらのファイルにアクセスする場合はパスワード認証を求めるようにします。ここでは一番簡単な「Basic 認証」で実現します。
まず、「.htaccess」ファイルをつくります。windows では「.」（ドット）で始まるファイル名は作成できないので、「dot.htaccess」という名前で作成します。

dot.htaccess

```
<Files ~ (inputBlog.php|deleteBlog.php)>
AuthType      Basic
AuthName    MembersOnly
AuthUserFile   (Full-Path-Name on Your Conputer )/.htpasswd
require  valid-user
</Files>
```

1 行目（と 6 行目）は、認証をかけるファイルを指定しています。
2 行目は認証方法として「Basic 認証」を使うという指定です。
3 行目の AuthName については何でもいいです。今回は「MembersOnly」としました。
5 行目はすべてのユーザに認証を求めるという設定です。
問題は、4 行目です。「.htpasswd」というのは、ユーザ名とパスワードを記録したファイルで、それがある場所をここに記述するのですが、「フルパス」で書かなければならないのです。たとえばあなたが xampp をお使いで、htdocs フォルダの中に「myblog」フォルダを作って、その中に今までに作成したファイル群を置いていて、.htaccess もその中に置くのであれば、おそらく以下のようになります。

```
C:/xampp/htdocs/myblog/.htpasswd
```

.htaccess と同じ場所に置かなければならないということはないので、別のところに設置してもかまいません。
パスワードの作成については、まず、パスワードを暗号化しなくてはなりません。パスワード生成して

くれるサイトを使えば.htpasswd に記述できる形式で表示してくれるので楽です。

**基本認証用パスワード暗号化**（ https://www.nishishi.com/scripts/htpasswd/ ）

他にも使いやすいサイトがあるはずです。

```
myname:dj7Z8bmD2XOMY
```

myname は、ユーザ名です。これを「dot.htpasswd」として保存します。
dot.htaccess と dot.htpasswd はそれぞれ、.htaccess .htpasswd というファイル名に変えておきます。
こののち、新規作成するとき、編集するとき、削除するときは、パスワードが求められるようになります。
（参考） Basic 認証(基本認証)でアクセス制限をかける方法（ https://allabout.co.jp/gm/gc/23780/ ）

### XAMPP の場合

ただ、XAMPP の場合は、XAMPP 専用のパスワード生成コマンドを使わねばなりません。それは、以下のところにあります。
C:¥xampp¥apache¥bin¥htpasswd.exe
コマンドプロントで、C:¥xampp¥apache¥bin フォルダに移動して、そこで、以下のようにすると、パスワードが生成されます。

```
> htpasswd -c dot.htpasswd myname
New password: ****
Re-type new password: ****
```

（注）****は、実際に入力したのを * で表現しています。（*を入力するわけではありません）
これで、dot.htpasswd に記録されているので、これを myblog-2 フォルダに置きます。そして、
「.htpasswd」
とファイル名を変更します。