session

Contents

- webの仕組み（復習）
- セッション機能
- 認証処理の実装
  - ログイン処理
  - ログアウト処理
- 課題発表 -> P2Pタイム

rules...

- 授業中は常にエディタを起動！
- 考えたことや感じたことはslackのガヤチャンネルでガンガン発信！
- 質問はslackへ！ 他の人の質問にも目を通そう！（同じ質問があるかも）
- 演習時，できた人はスクショなどslackに貼ってアウトプット！
- まずは打ち間違いを疑おう！
  - {} ' " ; など
- 書いたら保存しよう！（よく忘れる！）
  - command + s
  - ctrl + s

PHPの準備

以下3点ができているか確認しよう！

- XAMPPの起動確認
- http://localhost/のアクセス確認
- サンプルフォルダを「htdocs」フォルダに入れる

Goal

- ページ間でデータ共有する方法を知る！
- データの管理方法を学ぶ！
- ログイン&ログアウト処理を実装する！

前回の課題

【課題2】ユーザ管理機能の作成

ユーザ管理テーブル（<- 必ず作成，DBはこれまでのものを使用）

- テーブル名： users_table
- カラム名など

	#	名前	データ型	照合順序	属性	NULL	デフォルト値	コメント	その他	操作
☐	1	id	int(12)			いいえ	なし		AUTO_INCREMENT	変更 削除 その他
☐	2	username	varchar(128)	utf8mb4_unicode_ci		いいえ	なし			変更 削除 その他
☐	3	password	varchar(128)	utf8mb4_unicode_ci		いいえ	なし			変更 削除 その他
☐	4	is_admin	int(1)			いいえ	なし			変更 削除 その他
☐	5	is_deleted	int(1)			いいえ	なし			変更 削除 その他
☐	6	created_at	datetime			いいえ	なし			変更 削除 その他
☐	7	updated_at	datetime			いいえ	なし			変更 削除 その他

【課題2】ユーザ管理機能の作成

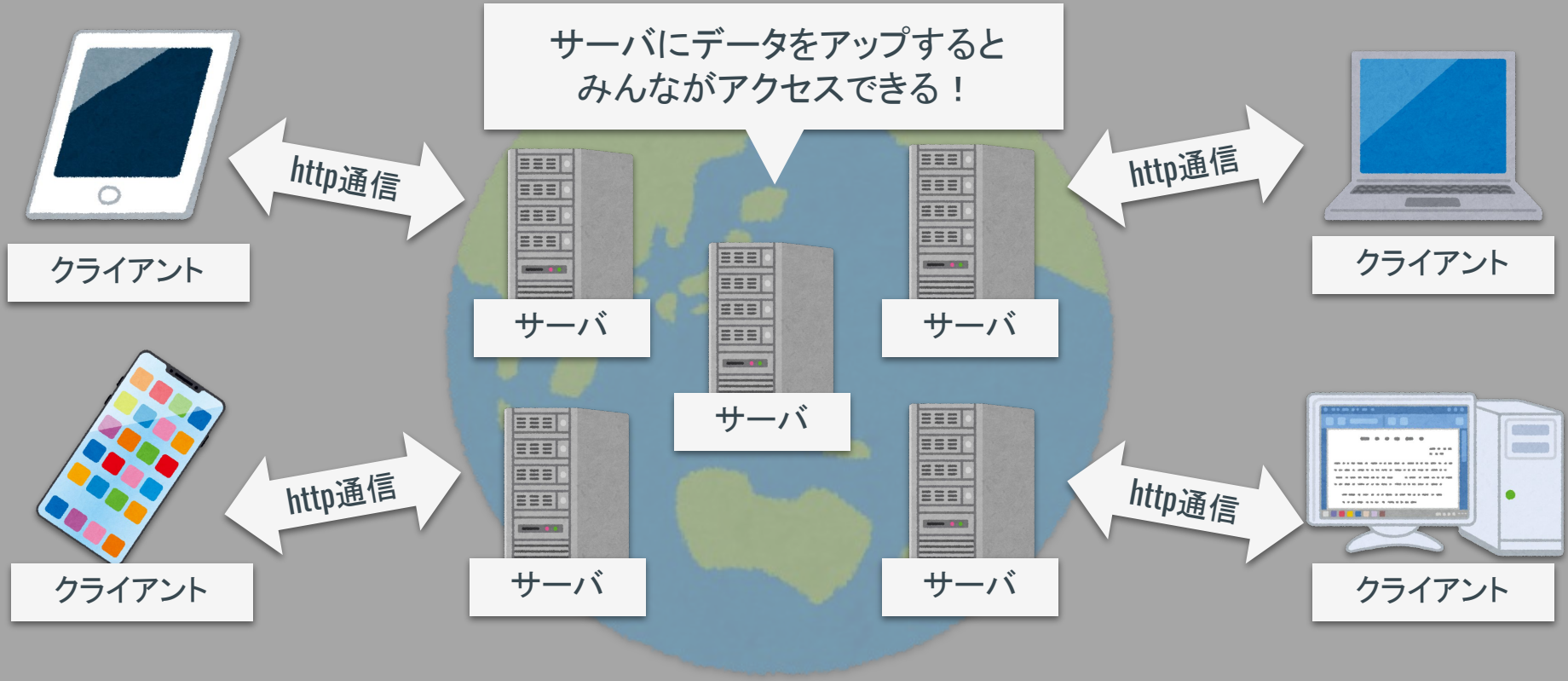
前スライドでつくったユーザのデータを管理する処理を実装！

- ユーザ作成処理
- ユーザー覧参照&表示処理
- ユーザデータ更新処理
- ユーザデータ削除処理
- （サービス管理者がユーザのデータを操作するイメージ）

課題1と課題2はそれぞれ独立でOK！

webの仕組み

雑なwebの仕組み

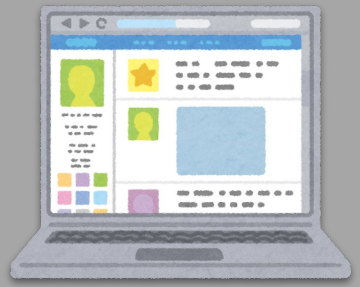

サーバにデータをアップすると
みんながアクセスできる！
http通信
http通信
http通信
http通信
クライアント
クライアント
クライアント
クライアント
サーバ
サーバ
サーバ
サーバ
サーバ

URL

- URLとは
  - web上にある情報（ファイル）の場所を指し示す住所.
  - Uniform Resource Locatorの略（覚えなくてOK）.
- 例

クライアントサイド言語の動き方

※ 言語によらず，ファイル（プログラム）はサーバ上に存在



送られてきたhtmlを実行

- こういう情報がほしい
- こういう処理をしたい
  例：index.htmlにアクセス

http通信

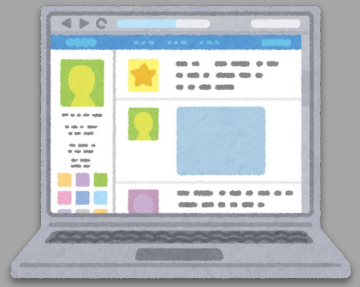
http通信

- リクエストしたhtml
- データを返すだけ！！

JavaScriptもこのタイミングで実行！！

DBの動き方

サーバ上のプログラム（PHPなど）がDBにアクセスして処理を実行！



送られてきたhtmlを実行

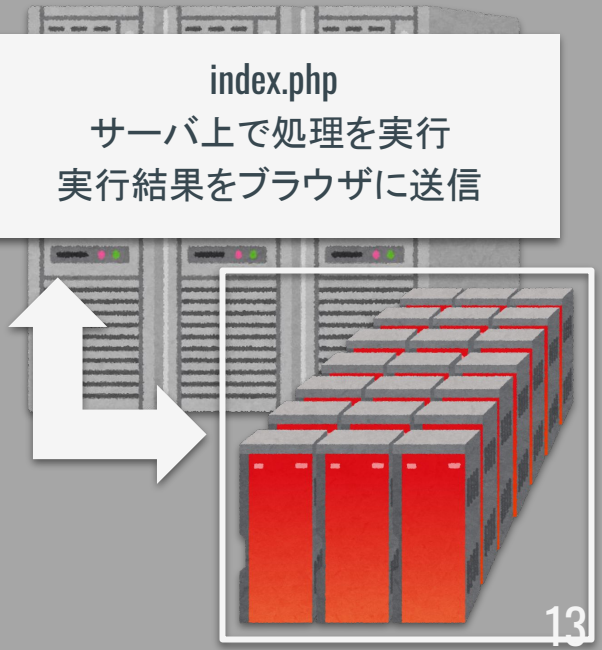
- こういう情報がほしい
- こういう処理をしたい
  例：index.phpにアクセス

http通信

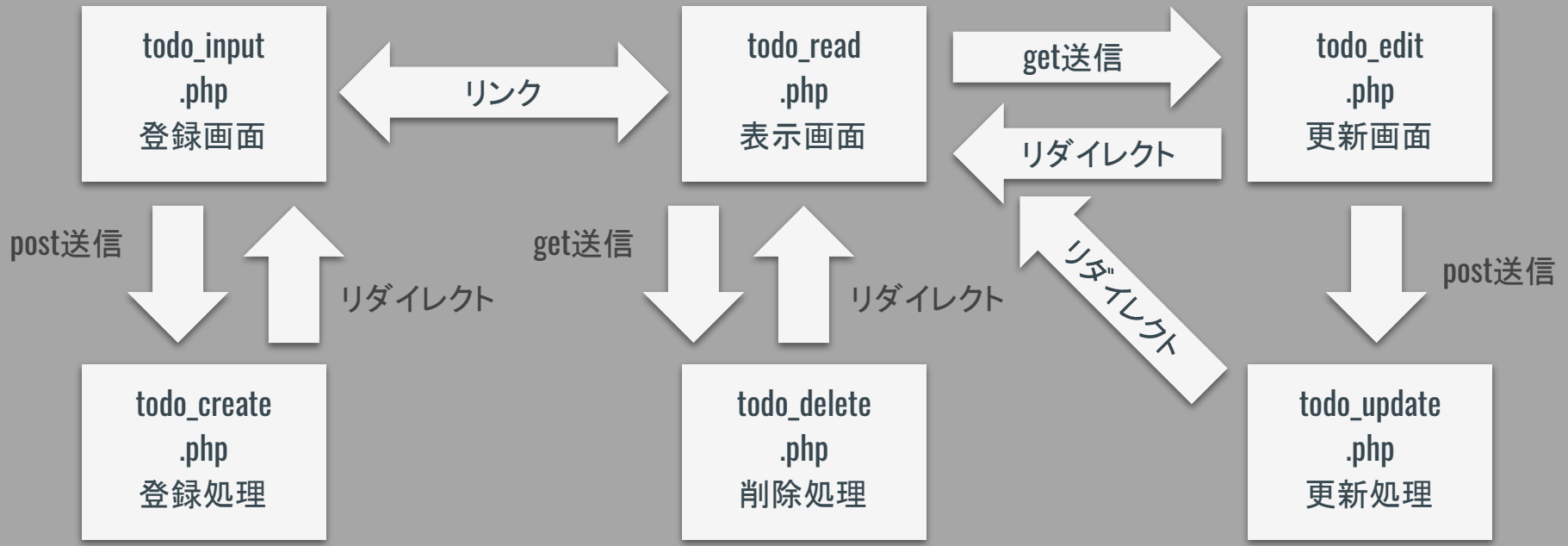
http通信

- 処理した結果のデータ
- 構成したhtml

index.php
サーバ上で処理を実行
実行結果をブラウザに送信



todoアプリの全体像



エラー表示用の設定

エラー表示用の設定

PHPではデフォルトでエラーが表示されない設定となっている.

開発時, エラーメッセージを確認しやすいよう設定の上書きを行う.

1. プロジェクトのフォルダ内に`.htaccess`ファイルを作成する.
2. `.htaccess`ファイルに下記の内容を追記して保存する.

```
php_flag display_errors On
```

これで, PHPのコードが実行されてエラーが発生した場合に画面で確認することができる.

セッション機能

sessionとは

- 何か？？
  - サーバに変数などを保存できる仕組み.
  - ……以上である！
- 補足
  - サーバ自体に変数を定義する.
  - サーバ上にあるどのファイルからでも値を取り出せる！

sessionとは

アプリケーションサーバ

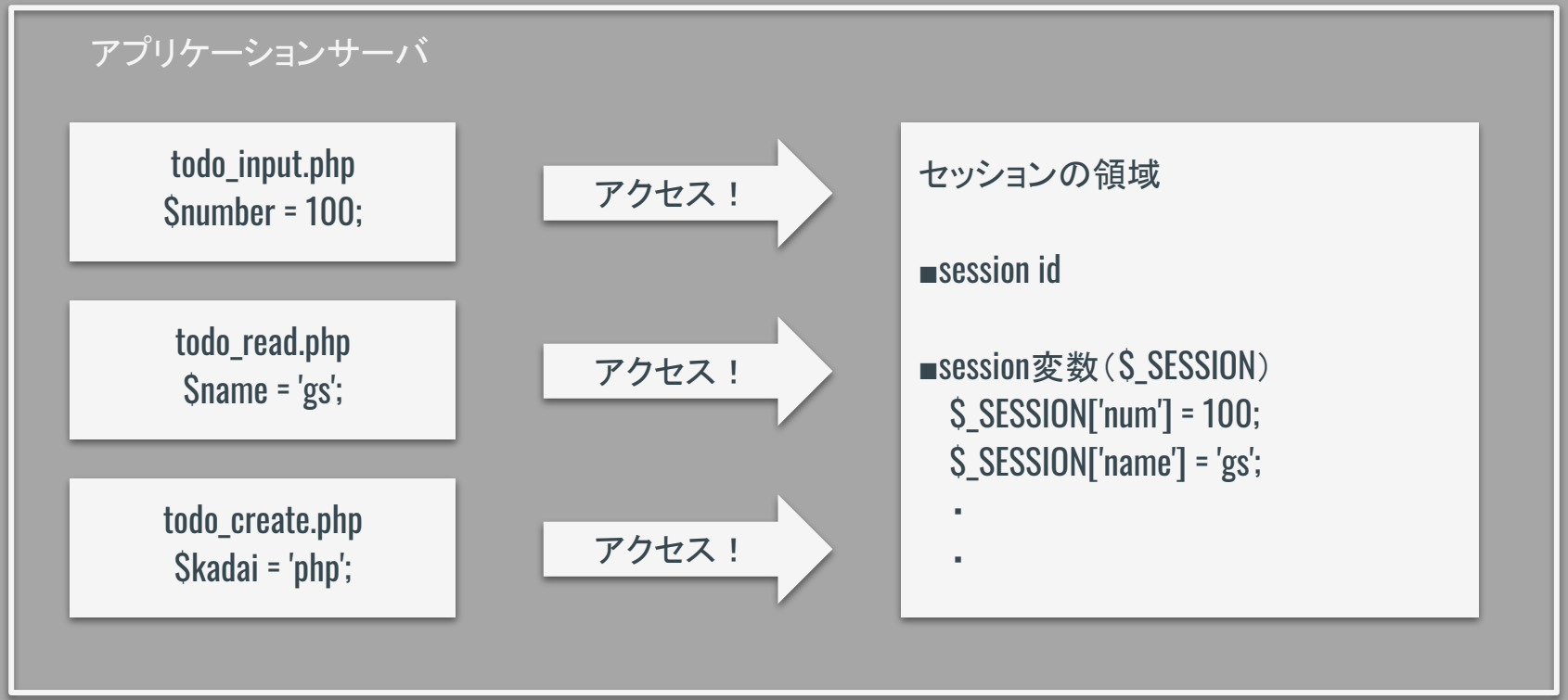
todo_input.php
$number = 100;

todo_read.php
$name = 'gs';

todo_create.php
$kadai = 'php';

アクセスできない！
変数はファイル内でのみ有効

sessionとは


アプリケーションサーバ
todo_input.php
$number = 100;
todo_read.php
$name = 'gs';
todo_create.php
$kadai = 'php';
アクセス！
アクセス！
アクセス！
セッションの領域
■session id
■session変数（$_SESSION）
$_SESSION['num'] = 100;
$_SESSION['name'] = 'gs';

session_id

session_id

sessionを始める

- sessionがスタートすると「session領域」が作られる.
- session領域識別用のid（session_id）が発行される.
- sessionの機能が使えるようになり，情報を保存できる.

session_idの再生成ができる

- 悪意あるサイトにsession_idを読まれてしまうとハッキングのリスク.

sessionを終了する

- 保存されている情報などを破棄する.

sessionを使う流れ（例）

sessionのスタート

- session_idの管理，必要に応じて$_SESSION（session変数）にデータを保存.

session_idの再生成

- ページ移動などのタイミングでidを更新する.

sessionの終了

session_idの確認

まずはsessionを宣言

- session_start();　<- 使用するファイルでは必ず最初に記述する！
- idが発行されてブラウザにidが保存される.

idの確認方法

- ①PHPファイル上でsession_id();で取得可能.
- ②ブラウザで「検証→Application→Cookies→localhost」

sessionの再生成

session_regenerate_id();

- sessionのidがバレると他の人にsessionの中身をいじられてしまう可能性. . ！
- session_regenerate_id();を使用するとidを再生成して更新できる.
- （保存されているデータ自体は変更なし）

使い所

- ログインしたらid発行してログイン情報を管理.
- ページ移動したタイミングで再生成&更新

sessionの再生成（session_regenerate_id.php）

```
// session_regenerate_id()の例

<?php
session_start();                                    // セッション開始
$old_session_id = session_id();                     // idの取得
session_regenerate_id(true);                        // id再生成&旧idを破棄
$new_session_id = session_id();                     // 新idの取得
echo '<p>旧id' . $old_session_id . '</p>';          // idの確認
echo '<p>新id' . $new_session_id . '</p>';
?>
```

session_idの管理と再生成（session_regenerate_id.php）

練習

- idを発行して確認しよう！
- 再生成して旧idと新idを表示しよう！

検証画面で確認（スライド23枚目）し，リロードの度に新IDと一致すればOK！

session変数

session変数に値を入れる！（session01.php）

```
// サーバに変数を保存する！
// $_SESSION['変数名']で宣言.

// 例
<?php
session_start();                    // session変数を使用する場合も必須！
$_SESSION['num'] = 100;             // session変数の宣言
echo $_SESSION['num'];
?>
```

session変数から値を取り出す！（session02.php）

```
// サーバに保存されている変数を取り出す.

// 例
<?php
session_start();                    // 必須！
$_SESSION['num'] += 1;              // session変数を+1する
echo $_SESSION['num'];              // 結果を出力
?>

// session02.phpでは変数を定義していないが，セッションの機能で呼び出せる！
```

session変数を扱う（session01.php / session02.php）

練習

- session01.phpでsession変数を定義しよう！
- session02.phpで定義した変数を呼び出して出力しよう！

sessionの終了

sessionの終了（情報の破棄）

```
// session変数の削除
unset($_SESSION[key]);          // 該当するsession変数を削除

// session情報の全削除
$_SESSION = array();            // session変数を空の配列で上書きする
setcookie(session_name(), '', time() - 42000, '/');
                                // ブラウザに保存した情報の有効期限を操作

// session領域自体をを破壊
session_destroy();

// 参考：https://www.php.net/manual/ja/function.session-destroy.php
```

認証処理

認証（ログイン&ログアウト）の全体像

必要なファイル

- todo_login.php　　ログイン情報（id, pwd）を入力して送信
- todo_login_act.php　　送信されたデータを受け取り，DB関連の処理を実行
- todo_logout.php　　セッション，ログイン情報の破棄

ログイン処理

ログイン処理の流れ

ログイン

1. ログインフォーム情報を入力して送信（todo_login.php）
2. 送信されたデータを受け取る（todo_login_act.php）
3. 受け取ったデータがDBにあるかどうかチェック（todo_login_act.php）

ログイン処理の流れ

成功時（DBにユーザのデータが存在した場合）

1. DBにログイン情報があればセッション変数に格納（todo_login_act.php）
2. セッション変数にログイン情報を保持してtodo_read.phpに移動

失敗時（DBにユーザのデータが存在しなかった場合）

- DBにログイン情報がなければtodo_login.phpに戻る（ログイン失敗）

ログインフォームの準備（todo_login.php）

```
<form action="todo_login_act.php" method="POST">
  ...       // actionとmethod
  <div>
    username: <input type="text" name="username">    // name属性
  </div>
  <div>
    password: <input type="text" name="password">    // name属性
  </div>
  <div>
    <button>Login</button>
  </div>
  ...
</form>
```

ログイン情報の受け取り（todo_login_act.php）

```
// セッション開始&ログイン情報の受け取り

<?php
session_start();                        // セッションの開始
include('functions.php');               // 関数ファイル読み込み
$pdo = connect_to_db();                     // DB接続
$username = $_POST[username];           // データ受け取り→変数に入れる
$password = $_POST['password'];
...

// 次ページへ続く
```

ログイン情報の検索（todo_login_act.php）

```
// DBにデータがあるかどうか検索

$sql = 'SELECT * FROM users_table
          WHERE username=:username
            AND password=:password
            AND is_deleted=0';

$stmt = $pdo->prepare($sql);
$stmt->bindValue(':username', $username, PDO::PARAM_STR);
$stmt->bindValue(':password', $password, PDO::PARAM_STR);
$status = $stmt->execute();

// 次ページへ続く（SQL失敗時のエラーなどは前回と同様の条件分岐）
```

WHEREで条件を指定！

ログイン情報の検索（todo_login_act.php）

```
// DBのデータ有無で条件分岐

$val = $stmt->fetch(PDO::FETCH_ASSOC);          // 該当レコードだけ取得
if (!$val) {        // 該当データがないときはログインページへのリンクを表示
  echo "<p>ログイン情報に誤りがあります. </p>";
  echo '<a href="todo_login.php">login</a>';
  exit();
}
...

// 次ページへ続く
```

ログイン情報をsession変数に保存（todo_login_act.php）

```
// DBにデータがあればセッション変数に格納
...
} else {
  $_SESSION = array();                          // セッション変数を空にする
  $_SESSION["session_id"] = session_id();
  $_SESSION["is_admin"] = $val["is_admin"];
  $_SESSION["username"] = $val["username"];
  header("Location:todo_read.php");             // 一覧ページへ移動
  exit();
}

// session変数には必要な値を保存する（今回は管理者フラグとユーザ名）.
// 自身のアプリで使いたい値を保存しましょう！
```

ログアウト処理

ログアウト処理（todo_logout.php）

```
// セッションの破棄→ログイン画面へ移動

<?php
session_start();                              // セッションの開始
$_SESSION = array();                          // セッション変数を空の配列で上書き
if (isset($_COOKIE[session_name()])) {
  setcookie(session_name(), '', time()-42000, '/');
}                                             // クッキーの保持期限を過去にする
session_destroy();                            // セッションの破棄
header('Location:todo_login.php');            // ログインページへ移動
exit();
```

ログイン / ログアウト処理を実装しよう！

ログインとログアウトの処理を実装しよう！

- ログインフォームを実装！（todo_login.php）
- ログイン処理を実装！（todo_login_act.php）
- ログアウト処理を実装！（todo_logout.php）

ログインフォームにidとpasswordを入力して一覧ページが表示されればOK！

ログイン状態の確認

ログインユーザのみの機能

ログインしているときのみアクセスできるように！

- 登録画面，一覧画面などはログイン済ユーザのみ見られるようにする
- ログインをチェックし，ログインしていない状態ならログイン画面に移動
- 「session_idを持っていない」or「idが古い」はログインしていない状態

複数のファイルでチェックを行うため，関数ファイルに記述しよう！

ログイン状態チェック関数（functions.php）

```
// ログインしているかどうかのチェック→毎回id再生成

function check_session_id () {
  // 失敗時はログイン画面に戻る
  if (!isset($_SESSION['session_id']) ||      // session_idがない
      $_SESSION['session_id'] != session_id()// idが一致しない
  ) {
      header('Location: todo_login.php');     // ログイン画面へ移動
  } else {
    session_regenerate_id(true);              // セッションidの再生成
    $_SESSION['session_id'] = session_id();   // セッション変数上書き
  }
}
```

ログインチェック関数の実行

```
// 各ページ読み込み時にログインチェック
<?php
session_start();                    // セッションの開始
include('functions.php');           // 関数ファイル読み込み
check_session_id();                 // idチェック関数の実行
...

// 下記ファイルで実施
// todo_input.php, todo_read.php, todo_edit.php, todo_create.php
// todo_create.php, todo_update.php, todo_delete.php
// ログインしていない状態でアクセスしようとするとログインページへ移動
```

ログインしていないとアクセスできないようにしよう！

メインのコンテンツは未ログインだとアクセスできないように！

- ログイン状態を確認する関数をつくろう！（functions.php）
- コンテンツのページで上記関数を実行しよう！
  - todo_read.php
  - todo_input.php
  - todo_create.php
  - todo_edit.php
  - todo_update.php
  - todo_delete.php

ログアウトした状態で各ページにアクセスできない状態になればOK！

課題

卒業制作プロトタイプ（継続）

今回はSessionの機能を使えるようになった！

- ユーザ管理
- ユーザデータとコンテンツのデータの連携
- ショッピングカート的なシステム

細かい部分の実装に使えるので工夫してみよう！

プロダクト全体の動線（ユーザがどう使うか）を考えてみるのがオススメ.

卒業制作プロダクト（仮）のプロトタイプを実装！！

卒業制作なにつくる？？

- 毎週機能を実装して進捗報告（発表あり！）.
- 講義内容に関連した機能を実装すると技術が定着する！

スケジュールつくろう！

- エクセルでも管理ツールでもなんでもOK！
- 作業進捗に応じて毎日修正しよう！
- 必要な「画面」「DB」「機能」をリストアップするのがコツ！
- （最初はうまくいかないのが普通なので，チャレンジしながら慣れる！！）

これまでのアプリケーションに認証処理を追加！

下記処理を追加しよう！（既存webサービスのコピー実装もオススメ！）

- ログイン画面
- ログイン処理
- ログアウト処理

ユーザの種類によって権限を分けてみよう

- ログインしていないユーザ　-> 情報を見るだけ（登録，更新&削除不可）
- 一般ユーザ　-> 情報の登録，表示，更新，削除が可能
- 管理者ユーザ　-> ユーザの登録，表示，更新，削除も可能

例（todoアプリの場合）

ログインしていないユーザがアクセス可能な画面

- ログイン画面
- 編集不可のtodo一覧画面

例（todoアプリの場合）

一般ユーザがアクセス可能な画面

- ログイン画面
- 編集不可のtodo一覧画面
- todo一覧画面
- todo登録画面
- todo編集画面

例（todoアプリの場合）

管理者ユーザがアクセス可能な画面

- ログイン画面
- 編集不可のtodo一覧画面
- todo一覧画面
- todo登録画面
- todo編集画面
- ユーザー覧画面
- ユーザ編集画面
- ユーザ編集画面

もうwebサービス作れるッ．．！

締切は厳守！！

P2Pタイム

まずはチーム内で解決を目指す！

訊かれた人は苦し紛れでも応える！！