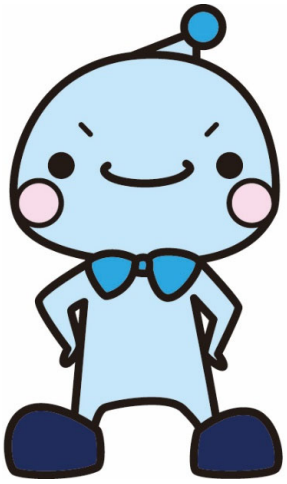
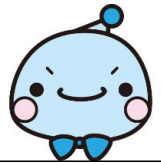
Python



盛岡情報ビジネス&デザイン専門学校

★☆★本日の内容★☆★

1. セッションとCookie
2. セッションの使い方
3. セッションの破棄
4. セッションの有効期限



◆ログイン処理の問題点

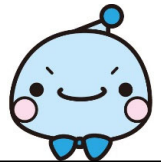
サンプルのログイン処理には問題点があります。

例えばログイン画面でID、PWを入力する代わりにURLに /mypage をつけてアクセスしてみましょう。

このように直接URLを入力することでログイン先のページを閲覧することができます。

このような攻撃手法を「強制ブラウジング」と言います。

これに対して、セッションを使って対策をしていきます。



◆セッションとCookie

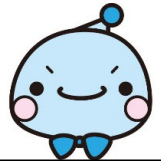
サーバ内のデータ格納領域です。

ここの領域にログイン成功時に情報を格納しておき、各ログイン後のコンテンツページにアクセスする際にセッションの情報を確認することで認証されているかどうかを確認することができます。

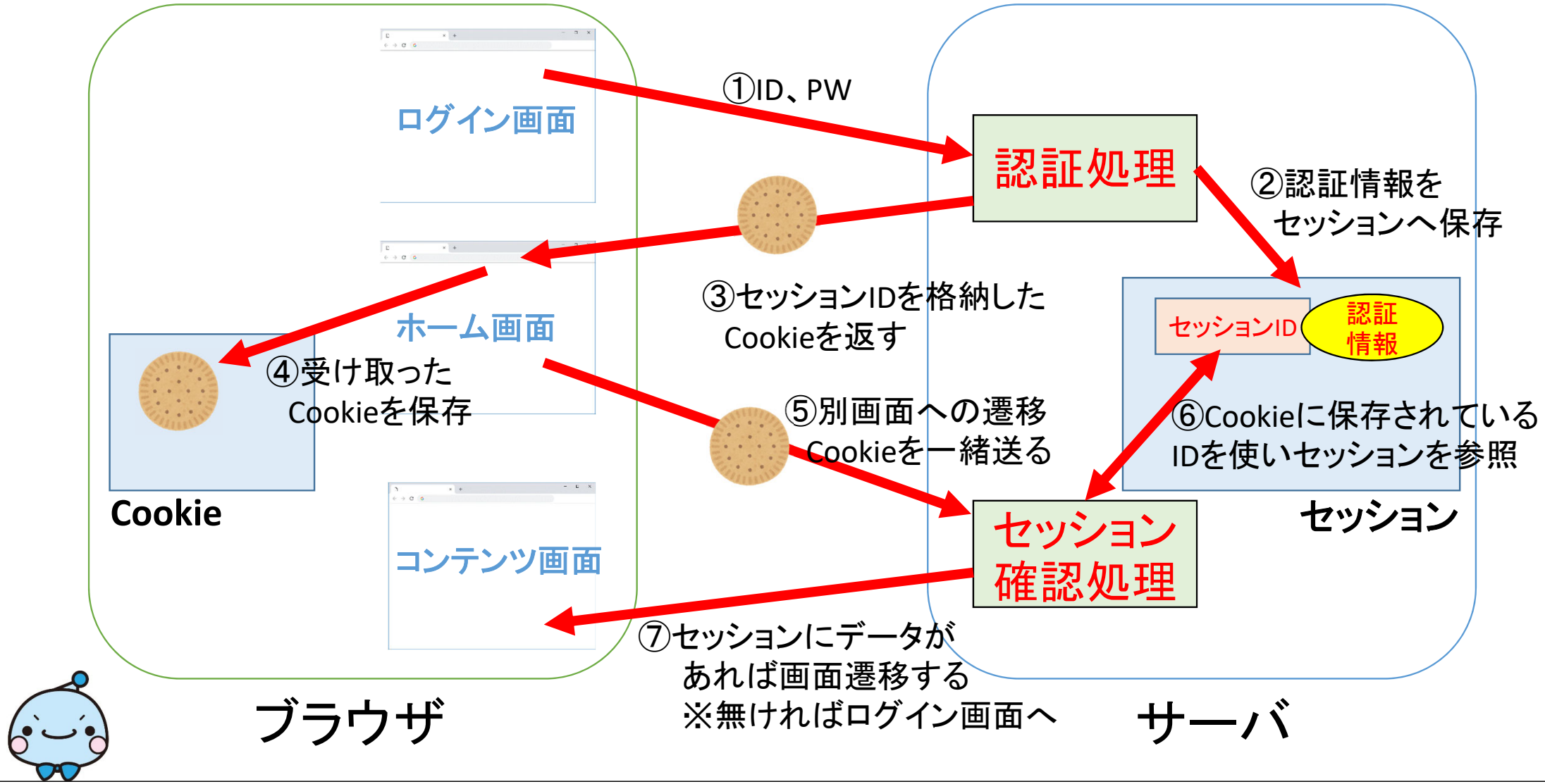
Cookieとはブラウザ上に保存されるデータのことです。

サーバはセッションに情報を保存した際にその情報に紐づくセッションIDをCookieに保存し、ブラウザへ渡します。

ブラウザはそのCookieを次のページに遷移する際にサーバに渡すことで、セッションに保存されたデータにアクセスすることができます。



◆セッションとCookie



◆セッションの使い方

Flaskでのセッションを使った認証処理を先ほどのログインのサンプルプログラムに追記していきましょう。

まずは必要なライブラリを追加していきます。

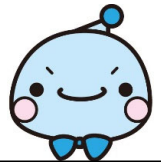
session はセッションを扱うためのライブラリです。

string と random は秘密鍵のためのランダムな文字列の生成に使用します。

次に Flask クラスのインスタンスに秘密鍵を設定します。
256桁のランダムな英文字列を設定しています。

```
from flask import Flask, render_template, request, redirect, url_for, session
import db, string, random

app = Flask(__name__)
app.secret_key = ''.join(random.choices(string.ascii_letters, k=256))
```

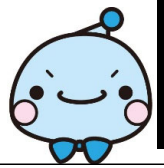


◆セッションの使い方

次にログイン成功時にセッションに情報を格納する処理を追記します。
Flaskのセッションは辞書型のように使用することができます。
今回の例ではキー：userに対してTrueという値を格納しています。
今回は論理値型を保存していますが、数値や文字列、リストなど様々な型のデータを保存することができます。

```
@app.route('/', methods=['POST'])
def login():
    user_name = request.form.get('username')
    password = request.form.get('password')

    # ログイン判定
    if db.login(user_name, password):
        session['user'] = True      # session にキー:'user', バリュー:True を追加
        return redirect(url_for('mypage'))
    else :
        error = 'ユーザ名またはパスワードが違います。'
        input_data = {'user_name':user_name, 'password':password}
        return render_template('index.html', error=error, data=input_data)
```

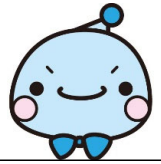


◆セッションの使い方

次に画面遷移時のログイン情報の確認処理です。

すべてのログイン後のコンテンツページにセッションの確認処理を実装する必要があります。

```
@app.route('/mypage', methods=['GET'])
def mypage():
    # session にキー:'user' があるか判定
    if 'user' in session:
        return render_template('mypage.html')   # session があれば mypage.html を表示
    else :
        return redirect(url_for('index'))   # session がなければログイン画面にリダイレクト
```



◆セッションの破棄

ログアウト時のセッションの破棄方法です。

popというメソッドで削除することができます。

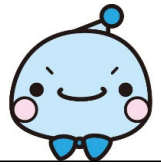
第1引数に対象のセッションのキーを指定します。

第2引数はもし、セッションが無かった場合の戻り値です。

対象セッションが存在する場合は戻り値は格納されているバリューです。

(今回の場合は True が返ります。変数に代入していないので、捨ててます)

```
@app.route('/logout')
def logout():
    session.pop('user', None)   # session の破棄
    return redirect(url_for('index'))   # ログイン画面にリダイレクト
```



◆セッションの有効期限

有効期限を設定する場合はセッション作成時に下記の処理を追加します。
値はシステムの要件や特性に応じて設定してあげる必要があります。

```python
@app.route('/', methods=['POST'])
def login():
    user_name = request.form.get('username')
    password = request.form.get('password')

    # ログイン判定
    if db.login(user_name, password):
        session['user'] = True      # session にキー:'user', バリュー:True を追加
        session.permanent = True    # session の有効期限を有効化
        app.permanent_session_lifetime = timedelta(minutes=5)   # session の有効期限を 5 分に設定
        return redirect(url_for('mypage'))
    else :
        error = 'ユーザ名またはパスワードが違います。'
        input_data = {'user_name':user_name, 'password':password}
        return render_template('index.html', error=error, data=input_data)
```