Section

1-3 開発環境を構築しよう（仮想環境）

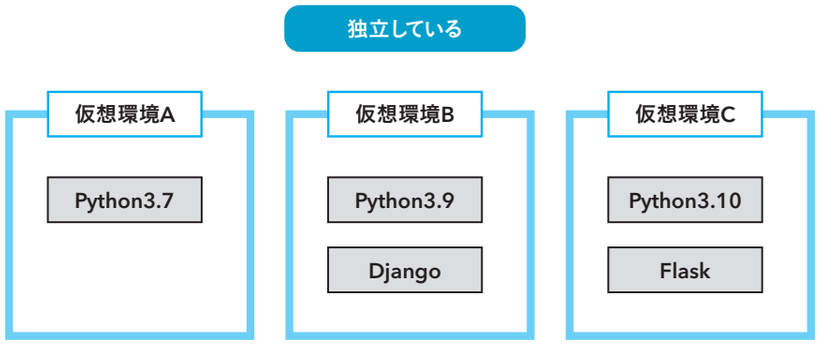
アプリケーション開発において、「プロジェクト」毎に必要な環境は異なります。たとえば、使用する「Pythonのバージョン」や必要な「パッケージやライブラリ」などです。プロジェクトが増えるたびに必要なパッケージやライブラリの種類も増えます。このような状況が続くと、管理できないほどパッケージの種類が増える、ライブラリ同士が干渉してプログラムが動かなくなるなどの問題が発生します。そこで、開発環境を管理するために「仮想環境」を使用することが推奨されます。

1-3-1 仮想環境とは？

プログラムにおける「環境」とは、プログラムを動かすために必要なソフトウェア群のことです。「仮想」とは、仮にあるものとして考えることです。「仮想環境」とは、同じPC内に複数の環境を仮想に構築することです。仮想環境は論理的に独立した環境で、パッケージによる依存性や互換性に左右されることがありません（**図1.14**）。

では、さっそく「仮想環境」を構築しましょう。

図1.14 仮想環境

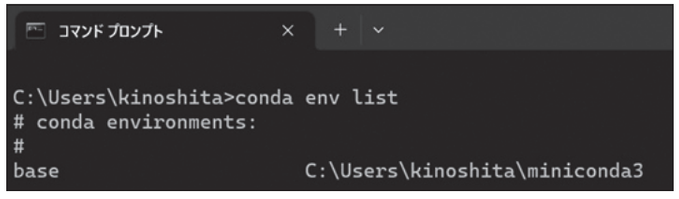


1-3-2 仮想環境の構築

仮想環境一覧

「コマンドプロンプト」の画面にて「conda env list」コマンドと入力後「Enter」キーをクリックします。仮想環境の一覧が表示されます（**図1.15**）。「base」は「Miniconda」が用意しているデフォルトの仮想環境です。

図1.15 仮想環境一覧



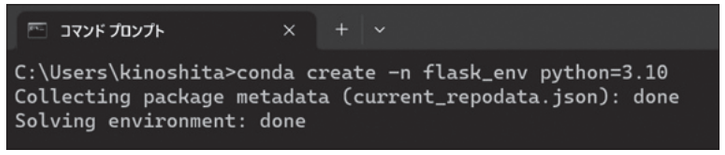
仮想環境の作成

仮想環境を作成するには以下のコマンドを実行します。

```
conda create -n [name] python=[version]
```

[name]は、自分で名づける仮想環境名、[version]はPythonのバージョンを指定できます。今回は仮想環境名「flask_env」、Pythonのバージョン「3.10」で仮想環境を作成しました。「conda create -n flask_env python=3.10」と入力し、仮想環境を作成します（**図1.16**）。

図1.16 仮想環境の作成



Proceed([y]/[n])? と表示されたら、「y」を入力後「Enter」キーをクリックします。これで「仮想環境」が作成されました。作成されたことを「conda env list」コマンドで確認します（**図1.17**）。

「flask_env」仮想環境が作成されたことを確認できます。

Section

2-1 Flaskでハローワールドを作成しよう

Flaskの具体的な説明に入る前に、プログラム学習のはじめの一歩として、Flaskで「ハローワールド」を作成しましょう。自分で動くプログラムを作成してから、その仕組みについて学び、脳にインプットすることで、知識が定着しやすくなると個人的に思います。早速、最初のFlaskアプリケーション「ハローワールド」を作成しましょう。

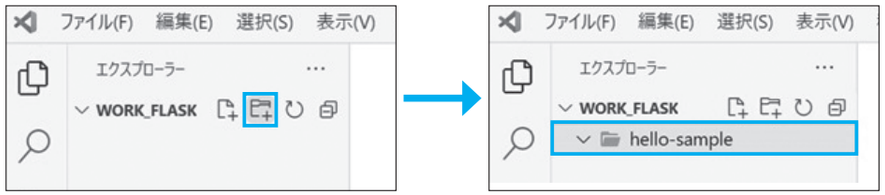
2-1-1 ハローワールドの作成

フォルダとファイルの作成

「1-4-3 ハンズオン環境の作成」で作成した「C:¥work_flask」ディレクトリに、今回作成するFlaskアプリ用のフォルダを作成します。

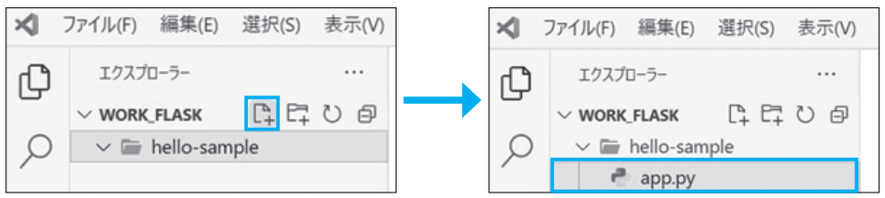
VSCode画面にて「新しいフォルダを作る」アイコンをクリックし、フォルダ「hello-sample」を作成します（**図2.1**）。

図2.1 フォルダの作成



作成したフォルダを選択後「新しいファイルを作る」アイコンをクリックし、ファイル「app.py」を作成します（**図2.2**）。

図2.2 ファイルの作成



Column サンプルファイルを利用する場合

「学習方法」としてお薦めする方法は、技術評論社の本書サポートページ（https://gihyo.jp/book/2023/978-4-297-13641-3/support）から、提供されている「リスト」をダウンロードして、ファイルに各リストを貼り付ける方法です。リストは動作確認済みです。まずはアプリケーションが動くことを確認し、その後、ご自身でコードについて学習することで、アプリケーションが動かないストレスから解放されます。「学習方法」の「効率的な方法」としてお話させて頂きました。

コードを書く

作成した「app.py」に**リスト2.1**のコードを記述します(注1)。Windowsの場合「Ctrl + S」キーを押すことでファイルを保存できます。またはVSCode画面ヘッダーにある、「ファイル→保存」でファイルを保存してください。

リスト2.1 app.py

```
001: from flask import Flask
002:
003: # ================================================
004: # インスタンス生成
005: # ================================================
006: app = Flask(__name__)
007:
008: # ================================================
009: # ルーティング
010: # ================================================
011: @app.route('/')
012: def hello_world():
013:     return '<h1>ハローワールド</h1>'
```

インタープリターを設定する

インタープリターとは、プログラムをPCが解釈・実行できる形式に変換しながら同時に少しずつ実行していくソフトウェアのことです。

VSCode画面で「app.py」を選択すると、画面右下に「Pythonインタープリター」が表示されます。もし、ご自身のVSCode画面でPythonインタープリターに「flask_env」仮想環境のPythonのバー

（注1） ソースコードの説明は、プログラム作成後に行います。

Section

3-1 テンプレートエンジンについて知ろう

この章では、Flaskのデフォルトの「テンプレートエンジン」である「Jinja2」について説明します。Flaskは「Jinja2」を組み込んでいるため、特別なインストール作業は必要ありません。豆知識として、読み方の由来は「テンプレート → テンプル → 神社（じんじゃ）」だそうです。まずは「テンプレートエンジン」について説明します。

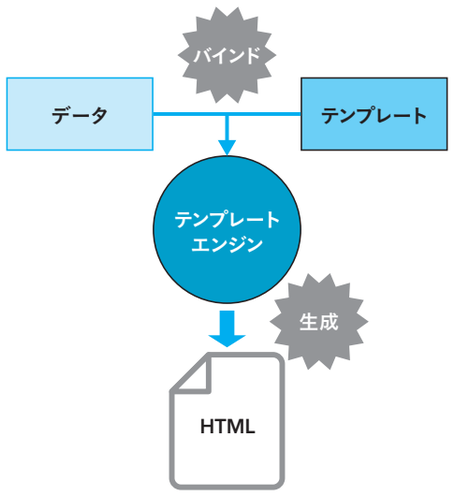
3-1-1 テンプレートエンジンとは？

「MVTモデル」のTに当たる「Template（テンプレート）」は、ユーザーに結果をどのように表示するかなど、結果データをもとにHTMLを生成してクライアントにレスポンスを返す役割を担います。

テンプレートエンジンとは、プログラム言語ごとにたくさんありますが、簡単に説明すると、「データとあらかじめ定義されたテンプレート（ひな型）をバインド（関連付け）して、Templateへの表示を助けるもの」です(注1)。

テンプレートエンジンを使用することで、Web開発者は同じデザインを使用して複数のページを生成することができます。テンプレートエンジンは、Webアプリケーション開発者にとって非常に便利なツールであり、効率的なWebアプリケーション開発を支援します（**図3.1**）。

図3.1 「テンプレートエンジン」のイメージ



（**注1**） バインドとは、何らかの要素やデータ、ファイルなどを相互に関連付けることを言います。

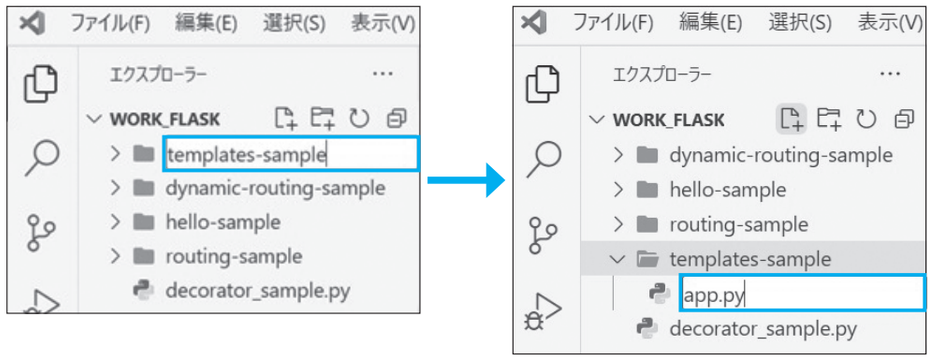
3-1-2 Jinja2を使用したプログラムの作成

フォルダとファイルの作成

「Jinja2」を使用したプログラムを作成して、より深く理解しましょう。

まず、「VSCode」の画面で「新しいフォルダを作る」アイコンをクリックし、フォルダ「templates-sample」を作成します。その後、作成したフォルダを選択し、「新しいファイルを作る」アイコンをクリックして、ファイル「app.py」を作成します（**図3.2**）。

図3.2 フォルダとファイルの作成



コードを書く

app.pyに**リスト3.1**のコードを記述します。

リスト3.1 app.py

```
001: from flask import Flask, render_template
002:
003: # ==============================================
004: # インスタンス生成
005: # ==============================================
006: app = Flask(__name__)
007:
008: # ==============================================
009: # ルーティング
010: # ==============================================
011: # TOPページ
012: @app.route('/')
013: def index():
014:     return render_template('top.html')
015:
016: # 一覧
017: @app.route('/list')
```

Section

4-3 エラーハンドリングを使おう

「エラーハンドリング」とは、プログラムの処理中に処理が妨げられる事象が発生した場合、その事象をエラーとして対処することを言います。Flaskでは「デコレーター」を使用することで簡単にエラーハンドリングができます。まずはエラーハンドリングに必要な前提知識を学習しましょう。

4-3-1 「HTTP ステータスコード」再び

パソコンやスマートフォンなどの「ブラウザ」からWebページを閲覧する時、**図4.10**の様に「サーバー」とのやり取りが行なわれています。

「サーバー」とはサービスを提供する方を指し、「クライアント」とはサービスを受ける方を指します（**図4.10**）。

「HTTP ステータスコード」とは、「HTTP レスポンス」に含まれる「サーバー」の「処理結果」を表す「3桁の数字」のことを指します（**図4.11**）。

図4.10 クライアントとサーバー

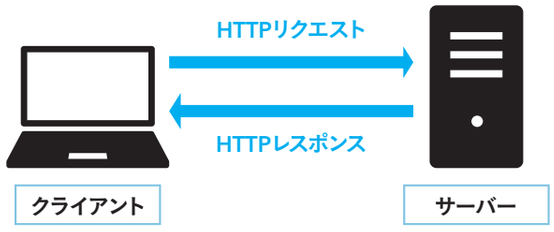
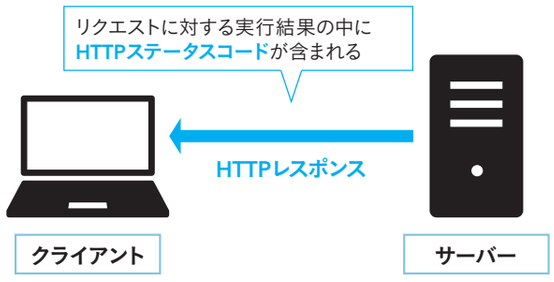


図4.11 HTTP レスポンス



「3桁の数字」は「200：リクエスト成功」、「404：ページが存在しない」などさまざまな意味を持っています。皆様も、1度は「404」を見たことがあるのではないでしょうか。

HTTP ステータスコードの分類を、**表4.2**に記述します。

表4.2 ステータスコードの分類

HTTPステータス（コード番号）	コードの内容	
100～	情報レスポンス	リクエストを受け、処理を継続
200～	成功レスポンス	リクエストに成功
300～	リダイレクション	リダイレクトなど、リクエストの完了には追加処理が必要
400～	クライアントエラー	クライアントからのリクエストに誤りがある
500～	サーバーエラー	サーバー側でリクエスト処理に失敗

4-3-2 「エラーハンドリング」の実装方法

Flaskで「エラーハンドリング」する方法は、以下のデコレーター「errorhandler」を使用します。

```
@app.errorhandler(ステータスコード)
def 関数名(引数):
```

4-3-3 「エラーハンドリング」を使用したプログラム作成

app.pyへの追加

プロジェクトtemplates-sampleのファイルapp.pyの「ルーティング」の続きに、**リスト4.8**を追記します。

リスト4.8 app.py

```
001: # エラーハンドリング
002: @app.errorhandler(404)
003: def show_404_page(error):
004:     msg = error.description
005:     print('エラー内容：',msg)
006:     return render_template('errors/404.html') , 404
```

2行目の「@app.errorhandler(404)」で「エラーハンドリング」を設定しています。ステータスコード「404」が発生した場合、関数「show_404_page」が呼ばれます。

6行目のreturnの末尾に「ステータスコード」を記述することで、ステータスコードを設定する

Section

5-3 Flask-WTFを使おう

WTFormsは、「フォーム」の作成、「バリデーション」、「フィールド」の自動生成などを行えました。実はFlaskとWTFormsの機能を組み合わせることで、Webアプリケーションの「フォーム」の作成、バリデーション、フィールドの自動生成などをより簡単に行える拡張機能があります。次は、Flaskアプリケーションで「フォーム」をさらに使いやすくするための拡張モジュール「Flask-WTF」について説明します。

5-3-1 Flask-WTFのインストール

「Flask-WTF」を使用するにはインストールする必要があります。仮想環境「flask_env」上でコマンド「pip install flask-wtf==1.1.1」と入力し「Flask-WTF」をインストールします（**図5.17**）。

図5.17 インストール

```
問題　出力　デバッグ コンソール　ターミナル　　cmd

(flask_env) C:\work_flask>pip install flask-wtf==1.1.1
Collecting flask-wtf==1.1.1
  Using cached Flask_WTF-1.1.1-py3-none-any.whl (12 kB)
Requirement already satisfied: Flask in c:\users\kinoshita\miniconda3\envs\flask_env\lib\site-packages (from fla
1.1) (2.3.2)
```

5-3-2 Flask-WTFの使用方法

フォルダとファイルの作成

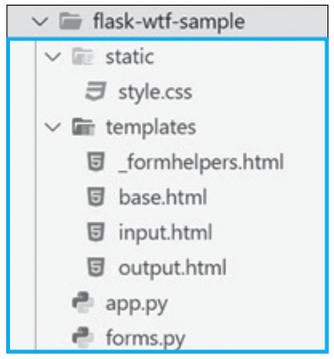
「Flask-WTF」を理解するために、実際にプログラムを作成しましょう。まず、VSCode画面で「新しいフォルダを作る」アイコンをクリックし、「flask-wtf-sample」という名前のフォルダを作成します。作成したフォルダを選択して、「新しいファイルを作る」アイコンをクリックし、「app.py」と「forms.py」を作成します。

次にフォルダ「flask-wtf-sample」配下に「新しいフォルダを作る」アイコンをクリックし、「templates」という名前のフォルダを作成します。フォルダ「templates」の中に、「base.html」と「input.html」と「output.html」と「_formhelpers.html」というファイルを作成します。

最後にフォルダ「flask-wtf-sample」配下に「新しいフォルダを作る」アイコンをクリックして、「static」という名前のフォルダを作成します。フォルダ「static」の中に、「style.css」という名前のファイルを作成します。

図5.18にフォルダとファイルの構造を示します。

図5.18 フォルダとファイルの作成



「Flask」の「static」フォルダは、静的ファイル（CSS、JavaScript、画像ファイルなど）を保存するためのディレクトリです。「static」フォルダ内のファイルは、Webアプリケーションからアクセスすることができます。

コードを書く

forms.py

ファイルforms.pyに、**リスト5.15**のコードを記述します。

リスト5.15 forms.py

```
001: from flask_wtf import FlaskForm
002: from wtforms import StringField, EmailField, SubmitField
003: from wtforms.validators import DataRequired, Email
004:
005: # ================================================
006: # Formクラス
007: # ================================================
008: # 入力クラス
009: class InputForm(FlaskForm):
010:     name = StringField('名前：', validators=[DataRequired('必須入力です')])
011:     email = EmailField('メールアドレス：',
012:                        validators=[Email('メールアドレスのフォーマットではありません')])
013:     submit = SubmitField('送信')
```

9行目「Flask-WTF」を使用する場合、「Form」クラスを作成するには「FlaskForm」クラスを継承したクラスを作成します。1行目で「FlaskForm」クラスを使用するために「import」しています。

Section 7-2

Flask-Migrateを使おう

「Flask-Migrate」はFlaskアプリケーションで「データベース」の「マイグレーション」を簡単に行うことができるライブラリです。このライブラリを使用することで、データベースのスキーマを変更する際に必要な手順を自動化することができます。「スキーマ」とは、データベースの構造や制約を定義する仕組みを指します。まずは「マイグレーション」について説明します。

7-2-1 マイグレーションとは？

「マイグレーション」とは、「データベース」の設計を変更する処理のことです。例えば、既存アプリケーションに新しい機能を追加する場合、「データベース」に新しいテーブルや列を追加する必要があります。その時、マイグレーションを行うことで、データベースの「スキーマ」を変更することができます。

マイグレーションの実行方法

マイグレーションの実行方法は、使用している言語やフレームワークによって異なりますが、一般的な方法は以下になります。

① マイグレーション用の「スクリプト」を作成します。
②「スクリプト」を実行して、データベースの「スキーマ」を変更します。

7-2-2 Flask-Migrateのインストール

Flaskでマイグレーションを使用するには「Flask-Migrate」を使用します。まずは「Flask-Migrate」を使用するためにインストールしましょう。仮想環境「flask_env」上でコマンド「pip install flask-migrate==4.0.4」と入力してFlask-Migrateをインストールします（**図7.4**）。

図7.4 Flask-Migrate

```
問題　出力　デバッグ コンソール　ターミナル

(flask_env) C:\work_flask>pip install flask-migrate==4.0.4
Collecting flask-migrate==4.0.4
  Using cached Flask_Migrate-4.0.4-py3-none-any.whl (20 kB)
Requirement already satisfied: alembic>=1.9.0 in c:\users\kinoshita\miniconda3\envs\flask_env\lib\sit
e-packages (from flask-migrate==4.0.4) (1.11.1)
Requirement already satisfied: Flask>=0.9 in c:\users\kinoshita\miniconda3\envs\flask_env\lib\site-pa
ckages (from flask-migrate==4.0.4) (2.3.2)
Requirement already satisfied: Flask-SQLAlchemy>=1.0 in c:\users\kinoshita\miniconda3\envs\flask_env\
lib\site-packages (from flask-migrate==4.0.4) (3.0.3)

cmd
Python
```

7-2-3 Flask-Migrateの使用方法

フォルダとファイルの作成

「Flask-Migrate」を使用したプログラムを作成し理解を深めましょう。VSCode画面にて「新しいフォルダを作る」アイコンをクリックし、フォルダ「flask-migrate-sample」を作成します。

作成したフォルダを選択後「新しいファイルを作る」アイコンをクリックし、ファイル「app.py」を作成します（**図7.5**）。

図7.5 フォルダとファイルの作成

app.pyへの書き込み

ファイルapp.pyに、**リスト7.2**を書き込みます。

リスト7.2 app.py

```
001: import os
002: from flask import Flask
003: from flask_sqlalchemy import SQLAlchemy
004: from flask_migrate import Migrate
005:
006: # ================================================
007: # インスタンス生成
008: # ================================================
009: app = Flask(__name__)
010:
011: # ================================================
012: # Flaskに対する設定
013: # ================================================
```

Section

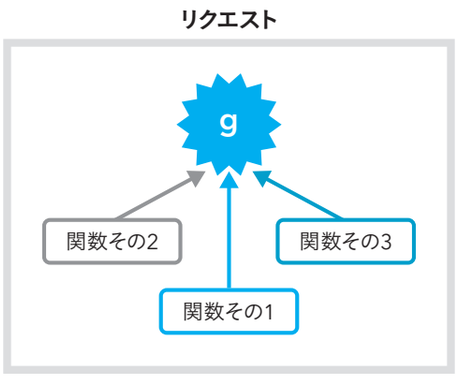
8-2 グローバル変数「g」を活用しよう

「グローバル変数」とは、プログラム内のどこからでもアクセスできる変数のことです。通常、プログラム内で変数を宣言すると、その変数はその変数を宣言したスコープ内でのみアクセス可能です。つまり、関数内で宣言された変数は、その関数内でのみアクセス可能であり、他の関数からはアクセスできません。しかし、グローバル変数を宣言することで、プログラム内のどこからでもその変数にアクセスできます。ここではFlaskのグローバル変数「g」について説明します。

8-2-1 グローバル変数「g」の概要

グローバル変数「g」は、Flaskアプリケーション内でデータを保存するために使用されます。リクエストの中で複数の関数にアクセスされる可能性のあるデータを格納することができます。グローバル変数「g」はリクエストが始まるときに作成され、リクエストが終わるときに破棄されます。リクエストの間だけ有効な一時的なデータを保存するために使用されます（**図8.4**）。

図8.4 グローバル変数「g」



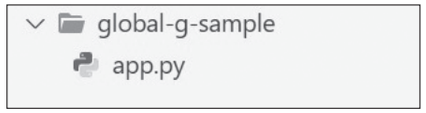
8-2-2 グローバル変数「g」を使用したアプリケーションの作成

フォルダとファイルの作成

グローバル変数「g」を使用したプログラムを作成し、理解を深めましょう。VSCode画面にて「新しいフォルダを作る」アイコンをクリックし、フォルダ「global-g-sample」を作成します。

作成したフォルダを選択後「新しいファイルを作る」アイコンをクリックし、ファイル「app.py」を作成します（**図8.5**）。

図8.5 フォルダとファイルの作成



app.pyへの書き込み

ファイルapp.pyに、**リスト8.10**を記述します。

リスト8.10 app.py

```
001: from flask import Flask, g, request
002:
003: # ================================================
004: # インスタンス生成
005: # ================================================
006: app = Flask(__name__)
007:
008: # ================================================
009: # ルーティング
010: # ================================================
011: @app.before_request
012: def before_request():
013:     g.user = get_user()
014:
015: @app.route('/')
016: def do_hello():
017:     user = g.user
018:     return f'こんにちは、{user}'
019:
020: @app.route('/morning')
021: def do_morning():
022:     user = g.user
023:     return f'おはようございます、{user}'
024:
025: @app.route('/evening')
026: def do_evening():
027:     user = g.user
028:     return f'こんばんは、{user}'
029:
030: # ユーザー情報を取得する処理
```

Section

11-1 認証処理の説明

この章では「イテレーション03：認証処理」の追加を作成していきましょう。Flaskにおいて、「ユーザー認証」と「セッション管理」を簡単に行うことができるライブラリ「Flask-Login」を使用して作成していきます。まずは「認証処理」の説明をします。

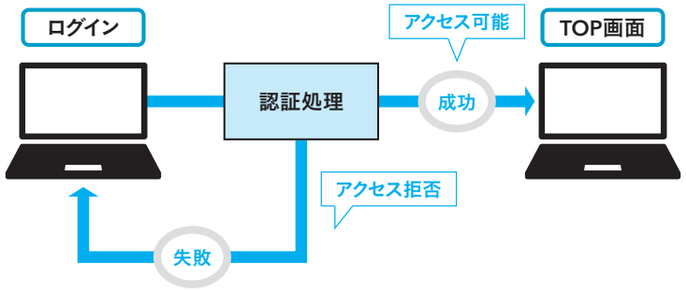
11-1-1 認証処理とは？

認証処理

「認証処理」とは、ユーザーが自分のアカウントにアクセスする際に、正しいユーザーかどうかを確認するプロセスのことです。一般的に、認証は「ユーザー名」と「パスワード」を使って行われます。

例えば、あなたがSNSやオンラインショッピングサイトにログインするとき、ユーザー名（またはメールアドレス）とパスワードを入力します。システムは入力された情報を使って、あなたが正しいユーザーかどうかをチェックし、認証が成功した場合、アカウントにアクセスできるようになります。認証が失敗した場合（例：ユーザー名またはパスワードが間違っているなど）、アクセスが拒否されます（**図11.1**）。

図11.1 認証処理



Flask-Login

「Flask-Login」は、Flaskにおいて、「ユーザー認証」と「セッション管理」を簡単に行うためのライブラリです。Flask-Loginを使用することで、セキュリティ性の高いWebアプリケーションを簡単に作成することができます。

Flask-Loginのインストール

コマンド「pip install flask-login==0.6.2」を実行し、インストールを行います（**図11.2**）。

図11.2 インストール

```
問題　出力　デバッグ コンソール　ターミナル

(flask_env) C:\work_flask\my_memo_app>pip install flask-login==0.6.2
Collecting flask-login==0.6.2
  Using cached Flask_Login-0.6.2-py3-none-any.whl (17 kB)
Requirement already satisfied: Flask>=1.0.4 in c:\users\kinoshita\miniconda3\envs\flask_env\lib\site-packa
ges (from flask-login==0.6.2) (2.3.2)

cmd
Python
```

Flask-Loginの主要機能

Flask-Loginの主要機能は以下6つになります。

- ユーザー認証状態の管理
  セッションを使用してログイン状態を保持し、リクエストごとにユーザーの認証状態を管理します。
- ユーザーのログイン処理
  「login_user()」関数を使用して、指定されたユーザーをログイン状態にします。
- ユーザーのログアウト処理
  「logout_user()」関数を使用して、現在ログインしているユーザーをログアウトさせます。
- ログインが必要なビュー関数への保護
  「@login_required」デコレーターを使用して、ログインが必要なビュー関数にアクセス制限をかけることができます。
- ログインページへのリダイレクト
  未認証のユーザーで「ログインが必要なビュー」にアクセスしようとした場合、自動的にログインページにリダイレクトさせることができます。
- current_user変数の提供
  Flask-Loginは「current_user」という変数を提供します。この変数を使用することで、現在ログインしているユーザー情報にアクセスすることができます。

「イテレーション03：認証処理」では上記の「Flask-Login」の主要機能を使用して作成していきます。

11-1-2 機能一覧

「イテレーション03：認証処理」の機能を**表11.1**に、URLに対する役割を**表11.2**に記述します。ルーティング処理時に使用しましょう。

Section

16-1 Bootstrapとは？

この章ではアプリケーション作成のラストミッションとして「イテレーション08：レイアウトを整える」を実施します。今回はウェブサイトを綺麗に見せるためのオープンソースのフレームワーク「Bootstrap」を利用します。

16-1-1 Bootstrapの概要

「Bootstrap（ブートストラップ）」は、Webサイトを綺麗に見せるためのオープンソースのフレームワークです。「Bootstrap」を使うと、スマートフォンやタブレット、パソコンなど、どんな画面でもきれいに見えるWebページが作れます。

特徴として、デバイスの画面サイズに応じて自動的にレイアウトが調整される「レスポンシブデザイン」や、ページを行と列に分割しコンテンツを整理しやすくする「グリッドシステム」などがあります。「Bootstrap」を利用することで初心者でも素早く、プロのようなデザインのウェブページを作成することが可能です。

本書では、「Bootstrap」について「レイアウトを整える」程度の説明しか行いません。詳細に知りたい場合は、別途参考書などの購入をお願い致します。また「Bootstrap」には「バージョン」があります。本書では「Bootstrap5」を使用します。「バージョン」によって使用できる記述方法が違うため、ご自身で「ネットで検索した正しい記述方法」なのにレイアウトにデザインが適応されないことが多々発生します。ご自身でネットで調べて「Bootstrap」を利用する場合は、「バージョン」による記述方法に気をつけてください。

- 公式Bootstrap v5.0
  https://getbootstrap.jp/docs/5.0/getting-started/introduction/

また、有志の方や公式サイトから「チートシート」が提供されています（**図16.1**）。「チートシート」とは、一般的にある技術に関する簡潔な情報や手順がまとめられた参照用資料（早見表）のことです。「チートシート」は、短時間でその技術の概要や使用方法を把握するのに役立ちます。「チートシート」を利用することで、様々なサイトを検索する手間が省け、効率的に「Bootstrap」を記述することが可能です。

図16.1 チートシート

チートシートとは、特定のトピックや技術に関する重要な情報を短く簡潔にまとめた**早見表**のようなものを指します

16-1-2 「Bootstrap5」の適用

新しくフォルダ構成などの作成はありません。早速「イテレーション08：レイアウトを整える」を実施しましょう。今回は、今まで作成した「MVT」の「T」にあたるテンプレートへ「Bootstrap」を追加することで課題をクリアできそうです。

「共通部分」の「T」への反映

templates/base.htmlを、**リスト16.1**に修正します。

リスト16.1 base.html

```
001: <!DOCTYPE html>
002: <html lang="ja">
003: <head>
004:     <meta charset="UTF-8">
005:     <meta name="viewport" content="width=device-width, initial-scale=1">
006:     <!-- Bootstrap CSS -->
007:     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.
min.css" rel="stylesheet" integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztc
QTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">
008:     <title>メモアプリ</title>
009: </head>
010: <body>
011:     <!-- ヘッダー部分に表示するナビゲーションバー -->
012:     {% if current_user.is_authenticated %}
013:         <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
014:             <div class="container">
015:                 <a class="navbar-brand" href="#">メモアプリ</a>
016:                 <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-
expanded="false" aria-label="Toggle navigation">
017:                     <span class="navbar-toggler-icon"></span>
```