Visual Studio Codeで始めるPythonプログラミング：

VS CodeとFlaskによるWebアプリ開発「最初の一歩」

http://www.atmarkit.co.jp/ait/articles/1807/24/news024.html [この記事はPDF出力に対応していません]

VS CodeとFlaskを組み合わせて、簡単なWebアプリを作りながら、ルーティング、テンプレートなどの基本を解説する。

2018年07月24日 05時00分 更新

[かわさきしんじ，Insider.NET編集部]

インデックス 連載「Visual Studio Codeで始めるPythonプログラミング」

FlaskはPythonでWebアプリを開発するための「マイクロ」フレームワークだ。今回から数回に分け、Visual Studio Code（以下、VS Code）とこのFlaskを利用して、Webアプリを開発するための基本を見ていこう。なお、本稿ではWindows版のVS CodeとPython 3.6.5を使用する（macOSでも動作を確認した）。

Flaskとは

FlaskはWebアプリを開発するための「マイクロ」フレームワークだ。ここで「マイクロ」とは単なる「小規模なフレームワーク」ではなく、「コア機能はシンプル」で「拡張性がある」ことを意味している。例えば、Flaskではデータベースアクセス機能やフォームの検証機能、ユーザー認証機能などは、コアな機能にはなく、拡張機能（extension）として提供されている。言い換えると、これは「開発者に多くの選択肢が残されている」ということであり、「どこに何を使用するか」を開発者が既存の拡張機能から自由に選べる（あるいは、自分で作成する）ことを意味している。

Flaskの公式サイト

Flaskの特徴（コア機能）としては、次のようなことが挙げられる。

- RESTfulなルーティング
- Jinja2テンプレートエンジン
- WSGI 1.0***1**に準拠
- 組み込みの開発用サーバを提供

***1** Web Server Gateway Interface。Pythonで記述されたWebアプリフレームワークと Pythonで記述されたWebアプリとを接続するためのインタフェースを仕様としてまとめたもの。本連載では表だって取り上げることはない。

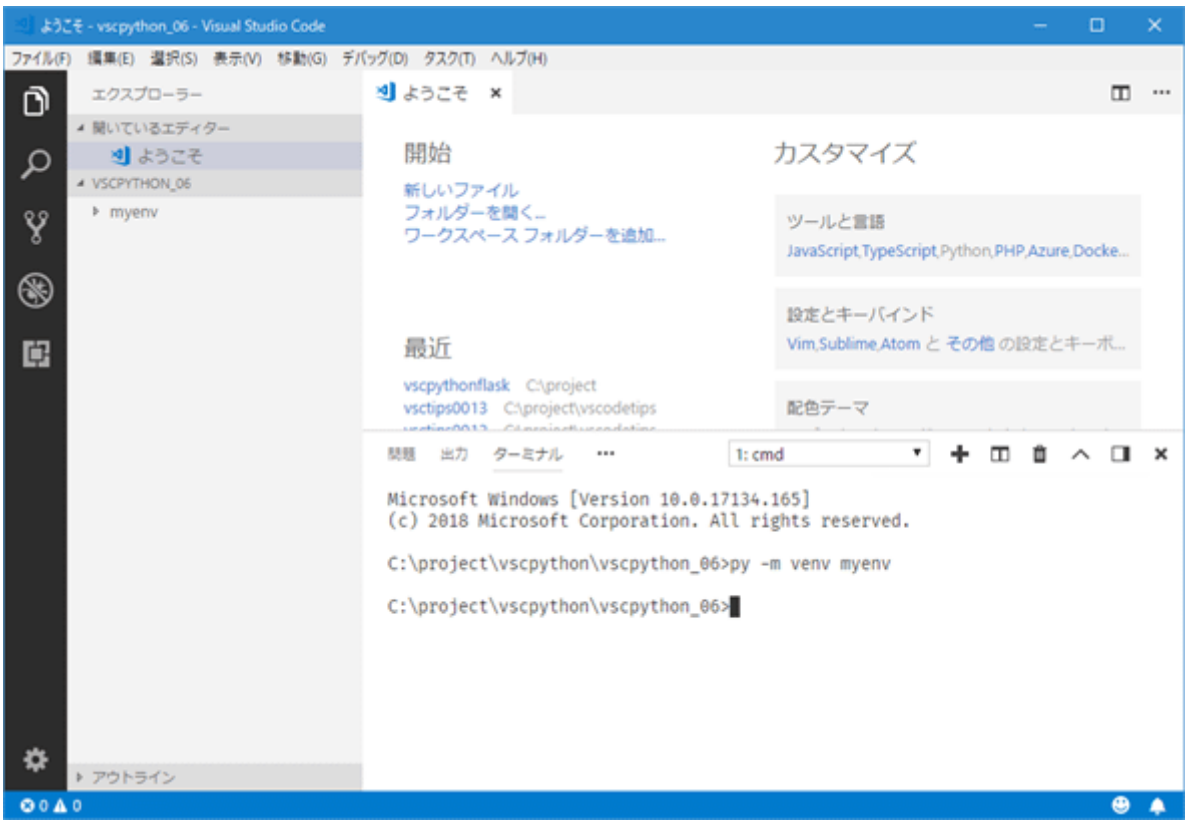
環境のセットアップ

本稿ではWindows版のVS Codeを利用して、以下の手順で開発環境をセットアップし、実際のコードを記述してみよう。

1. venvによる仮想環境の作成
2. Flaskのインストール
3. コードの記述
4. デバッグ実行

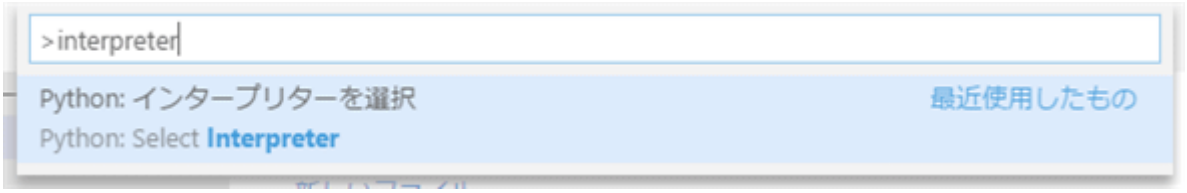
venvによる仮想環境の作成

venvによる仮想環境の作成は、WindowsであればVS Codeの統合ターミナルから「py -m venv <仮想環境名>」コマンドを実行するだけだ（Linux／macOSでは「python3 -m venv <仮想環境名>」コマンド。Python処理系の名前は「python3」ではなく、「python」かもしれないが、そこは読者の環境次第だ）。ここでは、VS Codeで適当なフォルダ（ここでは「vscpython_06」フォルダ）を開き、その下に仮想環境「myenv」を作成している。



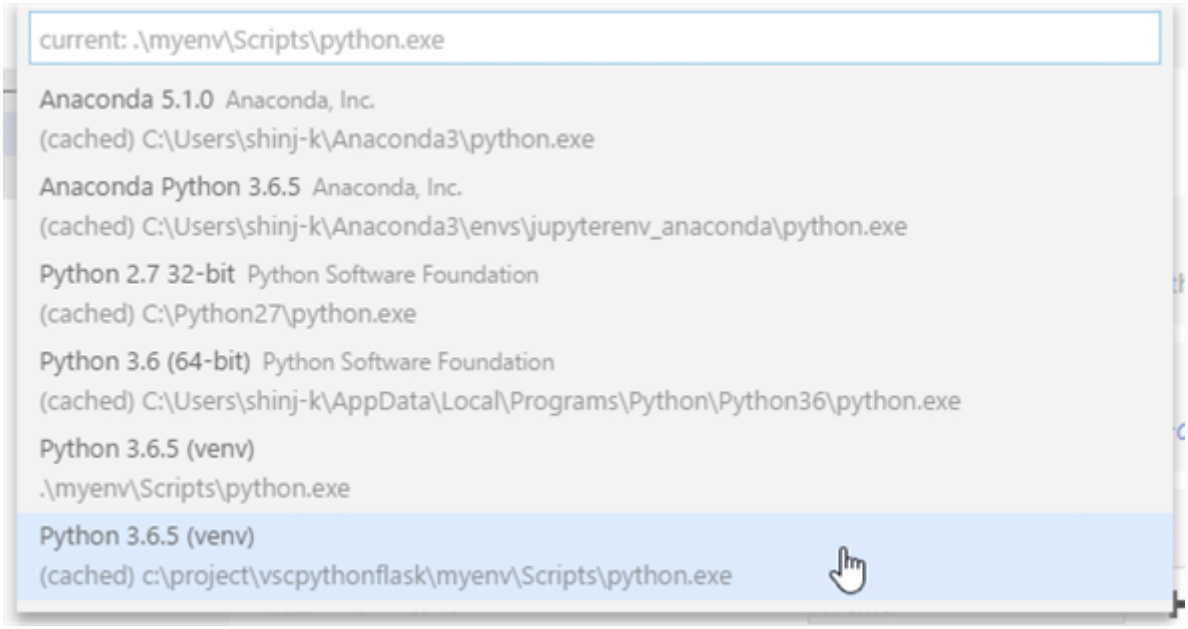
仮想環境「myenv」の作成

仮想環境を作成したら、コマンドパレットで「interpreter」などと入力して［Python: インタープリターを選択］コマンドを実行して、今作成した仮想環境を選択しておく（これにより、ワークスペース設定のpython.pythonPath項目の値が、作成した仮想環境に含まれるpythonコマンドに設定される）。



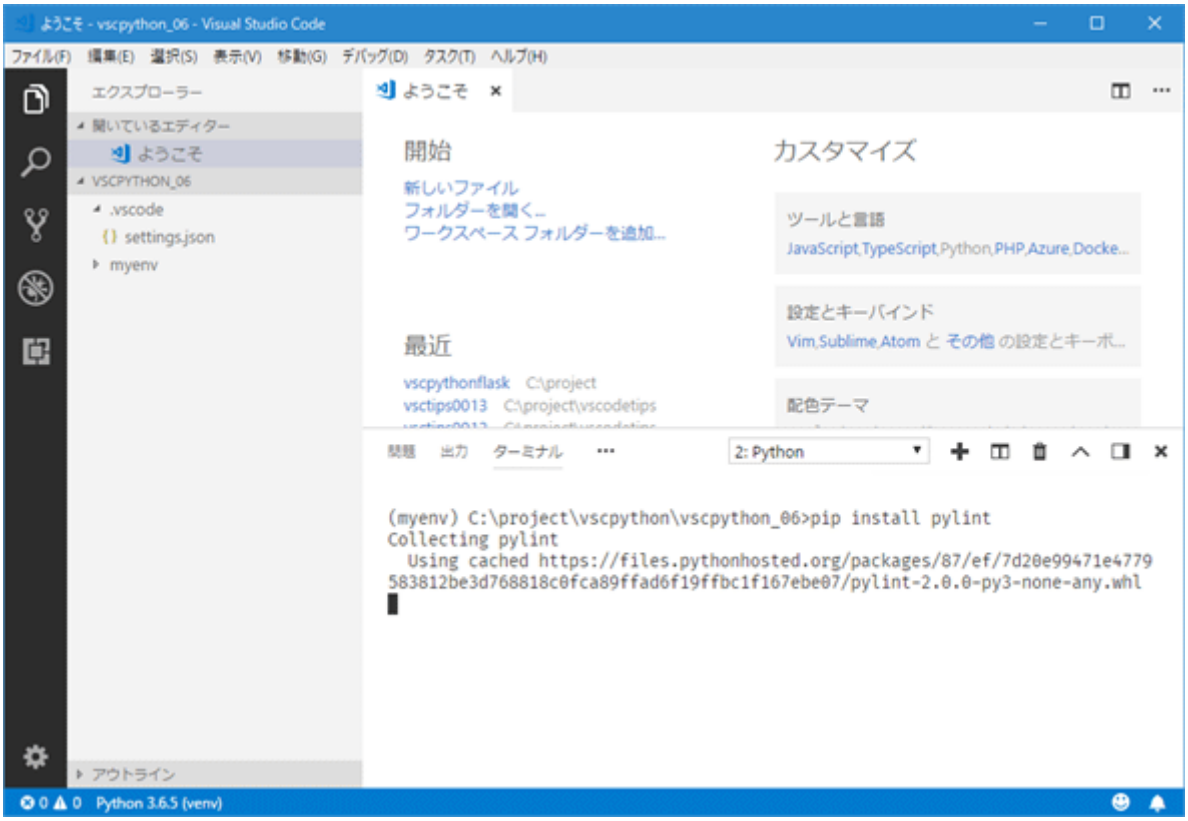
［Python: インタープリターを選択］コマンドを実行

▼



作成した仮想環境を選択
このフォルダ（ワークスペース）で使用するPython環境の選択

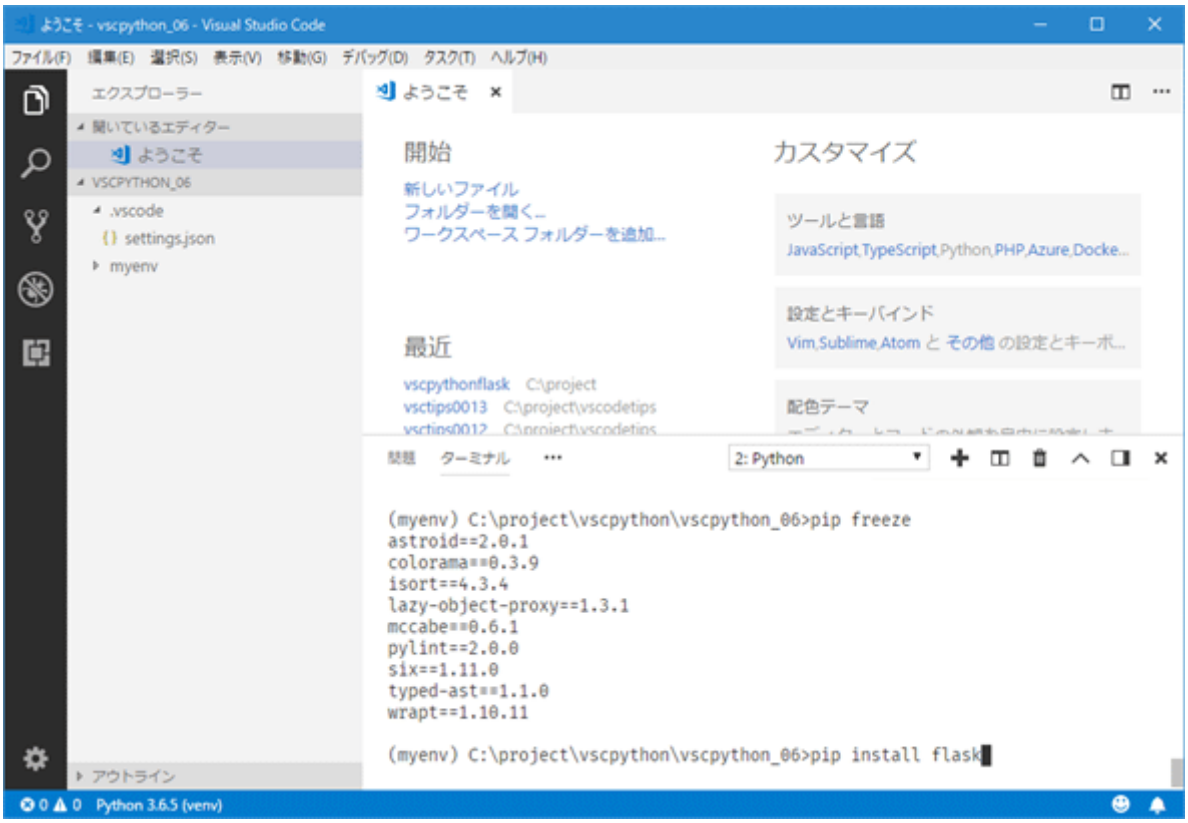
この後は、必要に応じてpylintのインストールなどをしておこう。このとき、仮想環境にパスが通ったターミナルを開くには、コマンドパレットの［Python: Create Terminal］コマンドを実行することを忘れないように。このコマンドを使えば、選択した仮想環境が最初から有効化された状態でターミナルを使用できる。



［Python: Create Terminal］コマンドで開いたターミナルでpylintをインストールしているところ

Flaskのインストール

Flaskをインストールするには「pip install flask」コマンドを実行するだけだ（上述した通り、コマンドパレットで［Python: Create Terminal］コマンドを実行して開いたターミナルを使おう）。



Flaskのインストール

上の画像では、Flaskのインストール前に「pip freeze」コマンドを実行して、仮想環境にインストール済みのパッケージを一覧している。以下はインストール後の同コマンドの実行結果だ。

Flaskインストール後の「pip freeze」コマンドの実行結果

Flaskに加えて、これが依存しているテンプレートエンジンの「Jinja2」やWSGIライブラリの「Werkzeug」、Flaskのコマンドラインインタフェースとなる「click」などもインストールされていることが分かる。

Flaskのインストールが完了したので、次ページでは簡単なFlaskアプリのコード例を見ながら、ルーティングやデバッグ方法について見ていこう。

Pythonコードの記述と実行

以下では、「最小のFlaskアプリ」を例に取り、Flaskアプリの基本構造、手作業でのアプリの実行、それを自動化してデバッグ実行を行う方法について見る。

コードの記述

以下に示すのは、公式サイトで紹介されている「最小のFlaskアプリ」だ。コードは「app.py」という名前のファイルに記述している。

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello_world():
  return "Hello, World!"
```

最小のFlaskアプリ（app.pyファイル）

最初の2行では、flaskパッケージからFlaskクラスをインポートして、そのインスタンスを作成している。気になるのは、インスタンス生成で使用されている「__name__」変数だ。

Pythonでは「__name__」変数の値は、Pythonモジュールが「スクリプトとして実行されている」か「他のコードでモジュールとして読み込まれているか」に応じて異なる。上記のapp.pyは、最終的には「flask run」コマンドあるいは「python -m flask run」コマンドを実行するときに環境変数FLASK_APPを介して、処理系に渡されるようになっている。この場合、__name__変数の値は「app」となる。一方、PythonのREPL環境で上記のコードを対話的に実行する場合などには、その値は「__main__」となる。実際には、Flaskはアプリで使用されるテンプレートや静的ファイルなどを探す際に、この値を利用するようになっている。

その後の行は、「最小のFlaskアプリ」のルーティング情報をapp.routeデコレータを利用して指示し、そのURL（ここでは「/」つまりアプリのルート）がリクエストされたときに、それを処理する関数を定義している。ここでは「Hello, World!」という文字列を返送するだけだ。

後はこれを実行するだけだ。VS Codeの［デバッグ］ビューを使えば、ブレークポイントでの中断なども行えるが、その前にこれを手作業で実行してみよう。

手作業でアプリを実行する

コマンドパレットの［Python: Create Terminal］コマンドで、仮想環境を有効化したターミナルを開き（既に開いていれば、それを利用して）、Windowsでは次のコマンドを実行する。「……」はプロンプトを省略したものだ（先頭に「(myenv)」を明示して、これが仮想環境であることを示している）。

```
(myenv) …… >set FLASK_APP=app.py
(myenv) …… >flask run
```

環境変数FLASK_APPにFlaskアプリをセットして、「flask run」コマンドを実行（Windows）

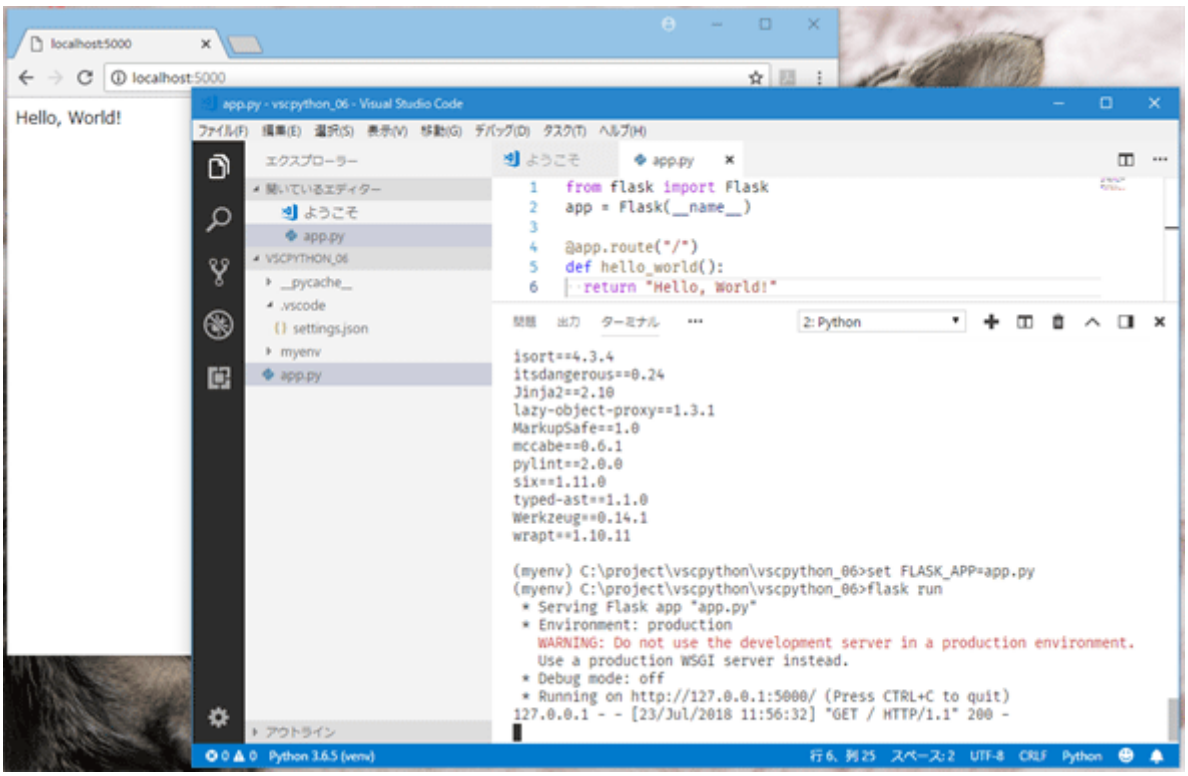
macOSやLinux（bash環境）なら、以下を実行する（ここではmacOSで動作を確認している）。

```
(myenv) …… $ export FLASK_APP=app.py
(myenv) …… $ flask run
```

環境変数FLASK_APPにFlaskアプリをセットして、「flask run」コマンドを実行（macOS）

要するに、環境変数FLASK_APPにFlaskアプリのエントリポイントとなるファイルを指定して、「flask run」コマンドを実行するだけだ（「python -m flask run」コマンドを実行してもよい）。

実際に実行した様子を以下に示す。



「最小のFlaskアプリ」の実行画面

ターミナルの出力を見ると、ローカルホストのポート5000をアプリが待ち受けていることが、ブラウザではそのURLにアクセスして「Hello, World!」が返送されていることが分かる。

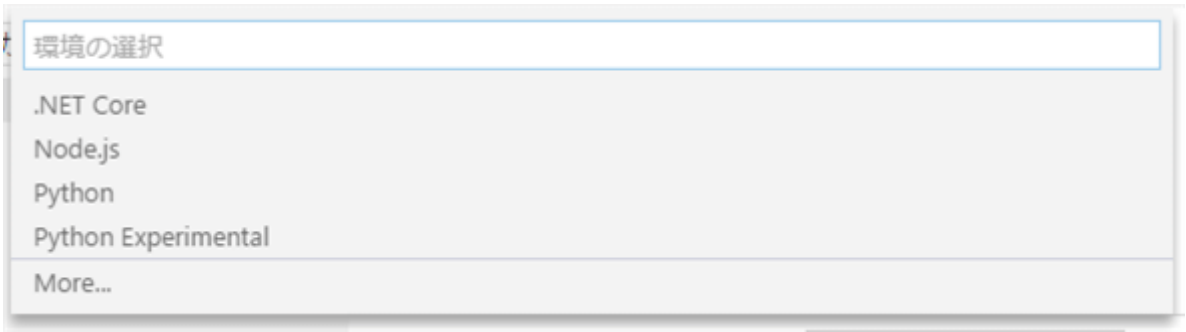
［Ctrl］＋［C］キーを押して実行を中断したら、今度はこれをデバッグ実行してみよう。なお、デバッグ実行については「VS CodeでPythonコードのデバッグも楽々！！」でも触れているので、詳しくはそちらを参照してほしい。

VS CodeでのFlaskアプリのデバッグ実行

VS Codeの［デバッグ］ビューを開くと、「構成がありません」と［デバッグの開始］ボタンの隣に表示されている。VS CodeのPython拡張機能にはPythonコードのデバッグ実行に必要なlaunch.jsonファイルの生成機能があるので、ここではそれを利用する。Pythonモジュールをエディタで開き、アクティブにした状態で、［デバッグ］ビューの歯車アイコンをクリックする。

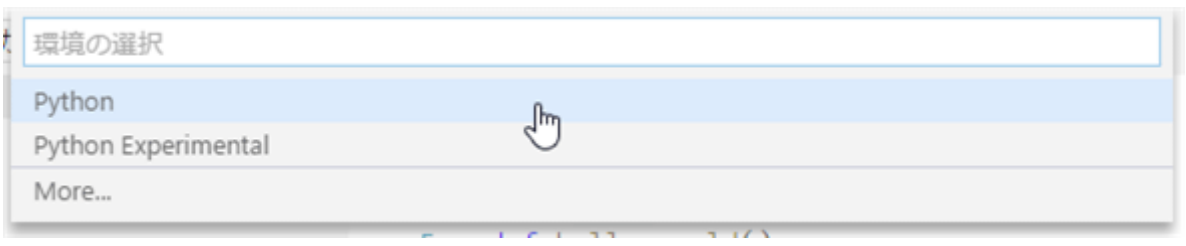
歯車アイコンをクリック

Pythonモジュールがアクティブでないと、次のようにどの環境に対応するlaunch.jsonファイルを生成するかを選択する一手間がかかるので注意しよう。



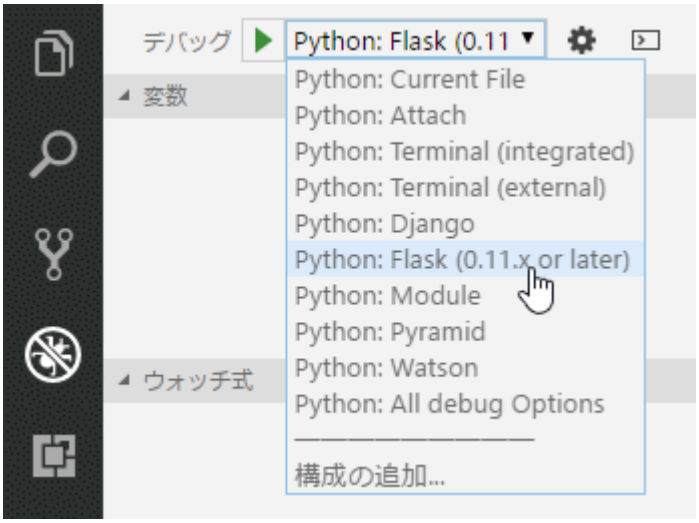
［環境の選択］メニュー

いずれにせよ、これにより［環境の選択］メニューが表示されるので、ここでは［Python］を選択しよう。



［環境の選択］メニュー

これにより、Pythonコードをデバッグするための各種構成が記述されたlaunch.jsonファイルが作成される。最後に［デバッグの開始］ボタンの隣にある、ドロップダウンから［Python: Flask］を選択すれば、デバッグ実行の準備は完了だ。



Flaskアプリのデバッグ用の構成を選択

ちなみにこの構成は次のようになっている。

```
{
 "version": "0.2.0",
 "configurations": [
  // …… 省略 ……
  {
   "name": "Python: Flask (0.11.x or later)",
   "type": "python",
   "request": "launch",
   "module": "flask",
   "env": {
    "FLASK_APP": "app.py"
   },
   "args": [
    "run",
    "--no-debugger",
    "--no-reload"
   ]
  },
  // …… 省略 ……
 ]
}
```

Flaskアプリをデバッグするための構成

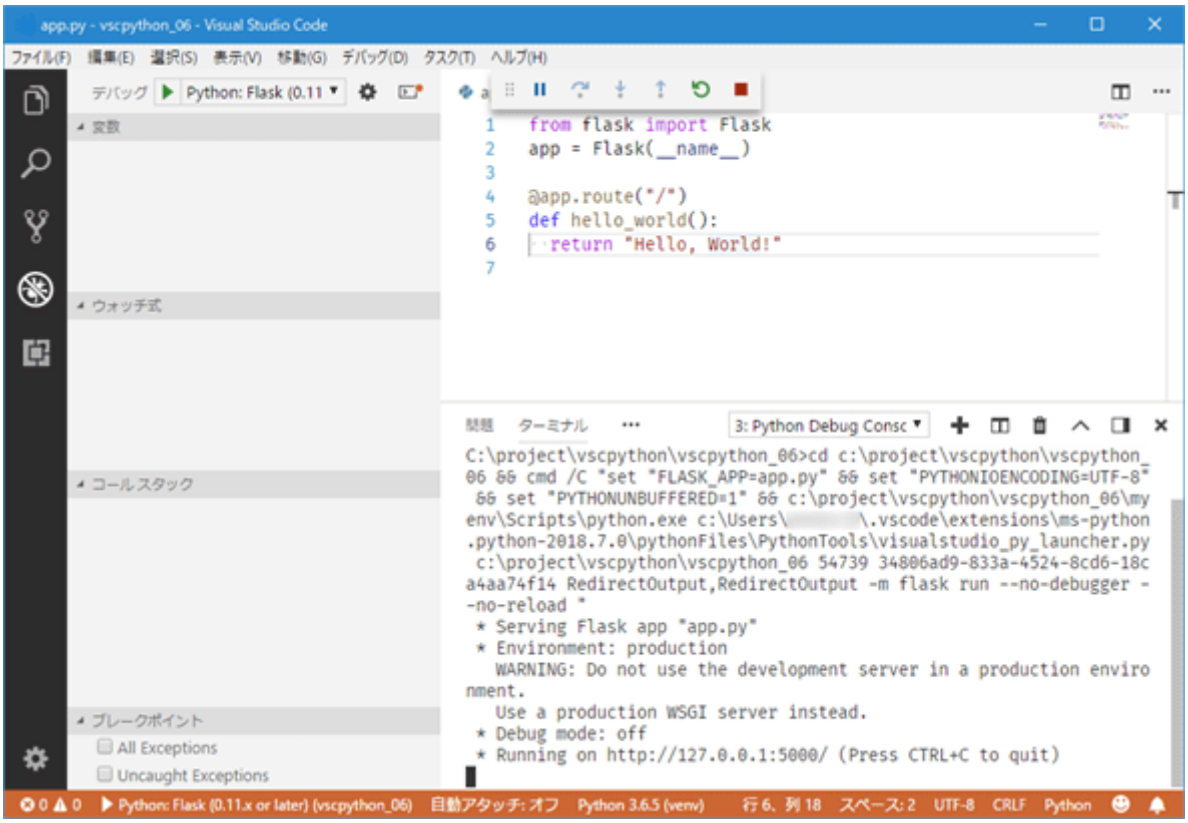
module項目はprogram項目で指定される「デバッグ実行開始時に起動するPythonファイル」を置き換えて、起動時に実行するPythonモジュールを指定するものだ（ただし、上に示した通り、ここではそもそもprogram項目は指定されていない）。ここでは、もちろん「flask」が指定されている（先ほどの「flask run」コマンドに相当）。

env項目は、デバッグ実行時に定義される環境変数を定義するもの。既に見た通り、ここでは環境変数FLASK_APPにアプリのエントリポイントとなるファイル（ここではapp.py）を指定して

いる（むしろ、この構成に合わせて、ファイル名をapp.pyとしている。必要であれば、FLASK_APPの値を変更しておこう）。

args項目はflaskコマンドに渡す引数となる。「run」はもちろん「flask run」コマンドに渡している「run」だ。「--no-debugger」はVS Codeのデバッガーと「Flaskが依存しているWerkzuegのデバッガーとの衝突を避けるため」の指定であり、「--no-reload」はVS CodeのPythonデバッガーがモジュールのリロードをサポートしていないために必要となっている。

［デバッグの開始］ボタンをクリックして、実際にデバッグ実行している様子を以下に示す。



デバッグを行っているところ

ここまでが、「最小のFlaskアプリ」を記述して、デバッグ実行するまでの手順となる。次に、このアプリにルーティング情報を追加して、他のURLのリクエストも処理できるようにしてみよう。

ルーティング情報の追加

ここでは次の2つのルーティング情報を追加してみよう。

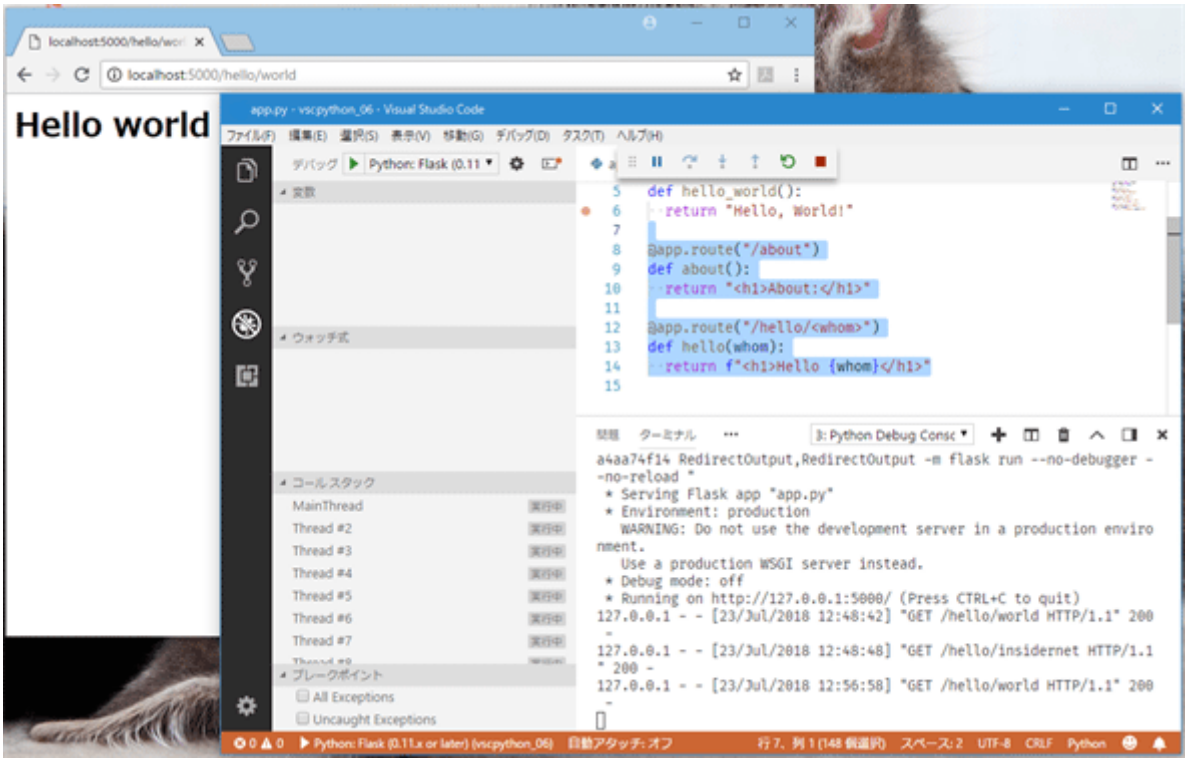
```
@app.route("/about")
def about():
  return "<h1>About:</h1>"

@app.route("/hello/<whom>")
def hello(whom):
  return f"<h1>Hello {whom}</h1>"
```

追加するルーティング情報

1つ目は単に「/about」がリクエストされたら、「<h1>About:</h1>」という文字列を返送するだけで、最初のものとあまり変わりはない。

2つ目はURLに「変数」を含めるようにしたものだ。ここでいう「変数」とは「<>」で囲まれた部分となる。つまり「/hello/world」というURLであればwhom変数の値が「world」に、「/hello/insidernet」であれば「insidernet」になる。これをhello関数ではwhomパラメーターに受け取る。ここではf文字列を使用して、その名前を埋め込んだ文字列を返送するようにしている。

デバッグ実行をして、後者のURLにアクセスしたところを以下に示す。



ローカルホストの「/hello/world」にアクセスしたところ

変数は「コンバーター」を使用して、特定の型であることを示すことも可能だ。以下に例を示す。

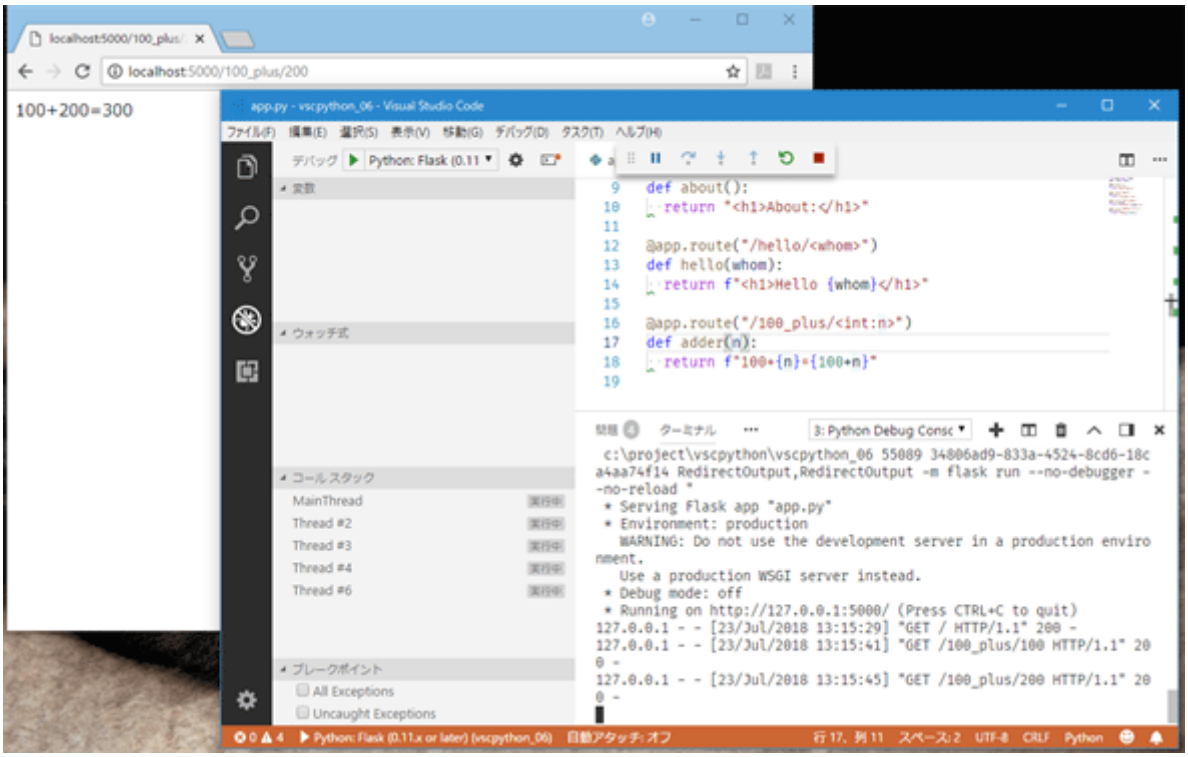
```
@app.route("/100_plus/<int:n>")
def adder(n):
  return f"100+{n}={100+n}"
```

変数nは整数

コンバーターは「<コンバーター:変数名>」のように指定する。使用できるコンバーターとしてはstring／int／float／path／uuidがある。これらを利用することで、リクエストを処理する際

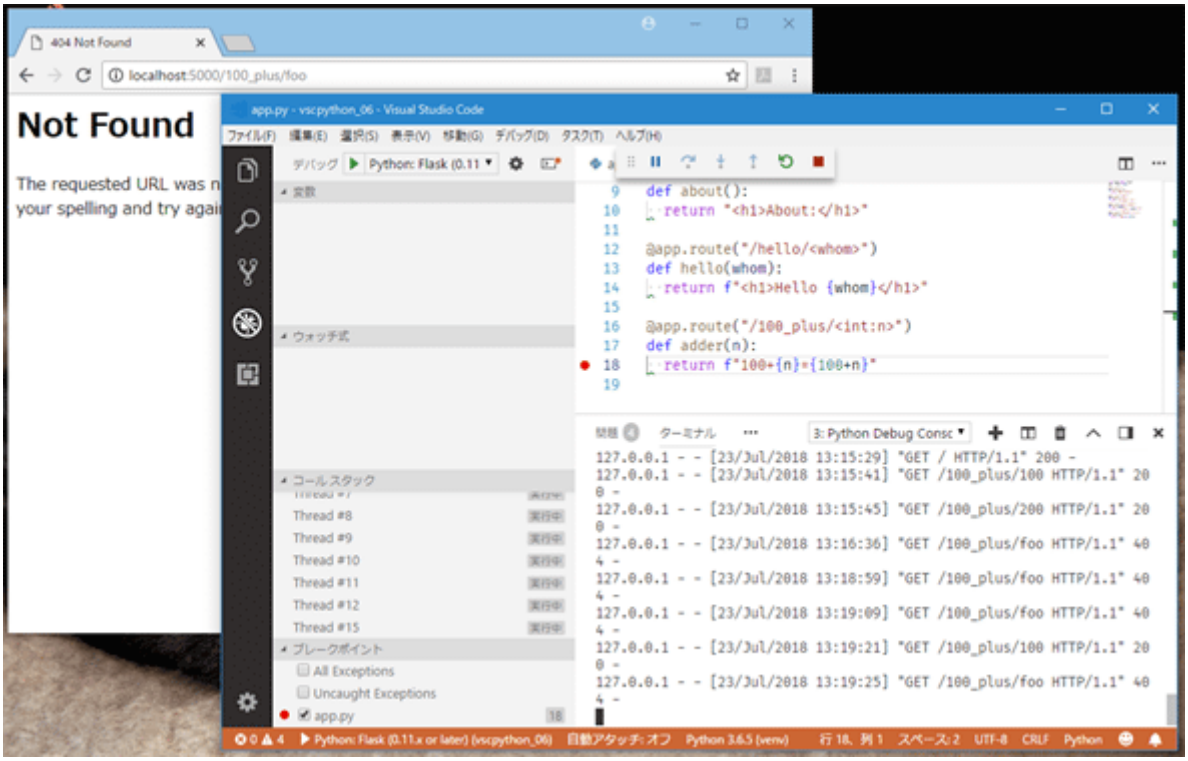
に文字列として受け取った値の処理が簡単になる。

以下は整数として受け取れる値をURLに含めた場合の例だ。



「100＋200」を計算して返送

整数として受け取れない値（foo）をURLに含めると次のようになる。



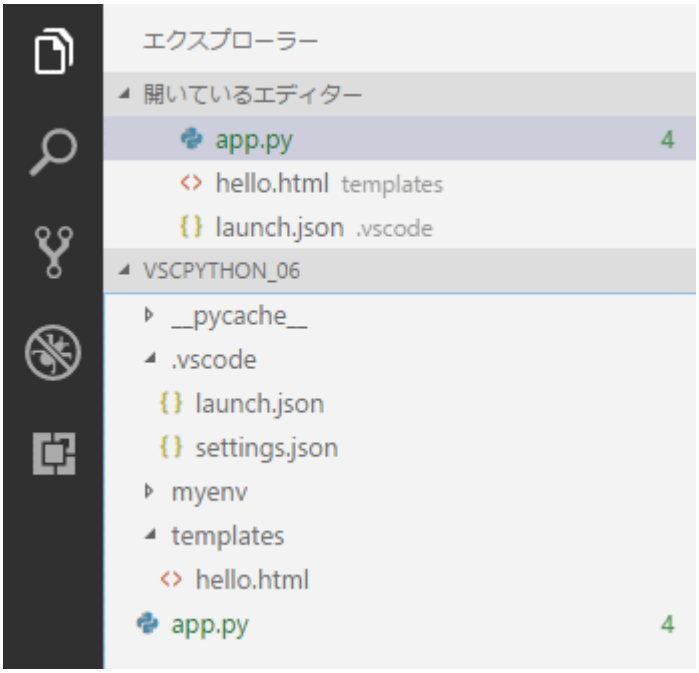
そんなURLは存在しない

VS Codeの側を見ると、ブレークポイントを設定していることが分かるが、これにヒットすることなく、エラーが返されている。コンバーターを使うことで、こうした処理を自前で実装する必要がないのは便利かもしれない。

Jinja2テンプレートエンジン

ここまでは、文字列あるいは文字列にHTMLタグを埋め込んだものを返送していたが、FlaskではWebページの描画に「Jinja2」というテンプレートエンジンを採用している。これを利用することで、生のHTMLではなく、より構造化されたレスポンスを返送できるようになる。「Flask最初の一歩」における最後の例として、これを使ってみよう。

Flaskでは、テンプレートをtemplatesフォルダから検索する。Flaskアプリがモジュール（単一ファイル）で構成されている場合には、templatesフォルダはモジュールと同じレベルに配置する。Flaskアプリがパッケージとして構成されているのであれば、パッケージ内に配置する。ここでは、app.pyファイル（モジュール）としてアプリが構成されているので、app.pyファイルと同じレベルにtemplatesフォルダを作成し、そこにhello.htmlファイルを配置しよう。



templatesフォルダをapp.pyファイルと同じ場所に作成

Jinja2の詳細な構文については「Template Designer Documentation」を参照してもらうとして、以下ではhello.htmlファイルに次のようなテンプレートを記述してみよう。

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <title>Hello</title>
</head>
<body>
```

```
  <h1>Using Jinja2 Template engine</h1>
  <h2>Hello {{whom}}</h2>
</body>
</html>
```

作成したテンプレート

ここで使用しているJinja2の構文は「<h2>Hello {{whom}}</h2>」の「{{ …… }}」という部分だけだ。これは、テンプレートエンジンに渡された変数の値を「{{ …… }}」内に描画するもの。これを呼び出して、Webページを描画するコードは次のようになる。

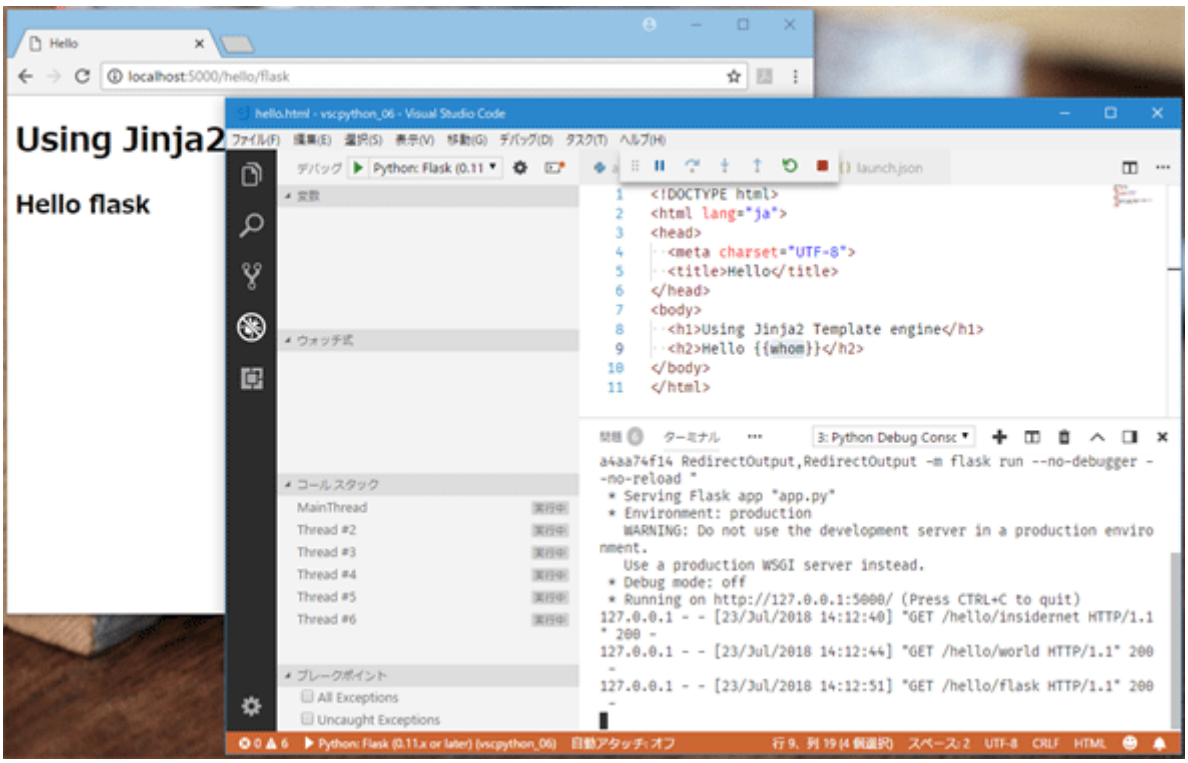
```
from flask import Flask, render_template
app = Flask(__name__)

# …… 省略 ……
@app.route("/hello/<whom>")
def hello(whom):
  return render_template("hello.html", whom=whom)

# …… 省略 ……
```

「/hello/<whom>」にアクセスすると、Jinja2テンプレートを利用して描画されたページが返送される

テンプレートを利用して描画したページを返送するには、render_template関数を利用する（これには、事前にrender_template関数をインポートする必要があることにも注意しよう）。

render_template関数の呼び出し時には、利用するテンプレートと、キーワード引数や辞書などを利用してテンプレート内で使用する変数を列挙する（上のコードではキーワード引数を使用）。

以下に修正後のアプリの実行画面を示す。



Jinja2テンプレートを利用したWebページの描画

テンプレートを利用してWebページが描画されたことが分かるはずだ。

今回は、VS CodeとFlaskを利用して、Webアプリ開発を行う「最初の一歩」として、仮想環境の作成、Flaskのインストール、ルーティング、テンプレート、VS Codeを利用したデバッグなどを取り上げた。次回はこれらの基礎を基にもう少し高度なアプリを作ってみることにしよう。

インデックス 「Visual Studio Codeで始めるPythonプログラミング」