
Visual Studio Code チュートリアル (Python)

usagi

2022 年 10 月 02 日

Contents

第 1 章	Visual Studio Code チュートリアル (Python 基礎)	3
1.1	Visual Studio Code と Python 拡張のインストール	3
1.2	Python のインストール	3
1.3	Python 仮想環境の構築と VS Code の起動	3
1.4	VS Code で使用する Python インタプリターを選択	4
1.5	Python ファイルの作成と実行	4
1.6	Python ファイルのデバッグ	5
1.7	パッケージの追加	5
1.8	グラフ描画用 Python ファイルの作成	6
1.9	Jupyter Notebook の拡張機能の利用	7
第 2 章	Visual Studio Code チュートリアル (Flask 基礎)	9
2.1	チュートリアル用の環境構築	9
2.2	最小限の Flask アプリの作成	10
2.3	アプリのデバッグ	11
2.4	テンプレートの利用	13
2.5	static ファイルの使用	14
2.6	ファイルの分割	16
2.7	積み残し	17

この資料は次の資料を参考に作成しています。

[Getting Started with Python in VS Code](#)

1. Visual Studio Code チュートリアル (Python 基礎)

1.1 Visual Studio Code と Python 拡張のインストール

1. Visual Studio Code をインストールします。
2. Python extensions for VS Code をインストールします。

注釈: Visual Studio Code を VS Code と略します。

1.2 Python のインストール

1. `python` から最新バージョンをダウンロードしてインストールします。
2. コマンドプロンプトを起動して Python がインストールされていることを確認します。

```
py -3 --version
```

注釈: `py` の代わりに `python` を使用することもできます。Windows では `py` が使えますが、Linux では `python` を使用してください。

1.3 Python 仮想環境の構築と VS Code の起動

Python を利用する際に様々なパッケージをインストールします。追加のパッケージをグローバル環境にインストールすると、他のプロジェクトにも影響するため、通常、Python の仮想環境内にインストールして使用します。

VS Code を起動したフォルダがプロジェクトのワークスペースになります。

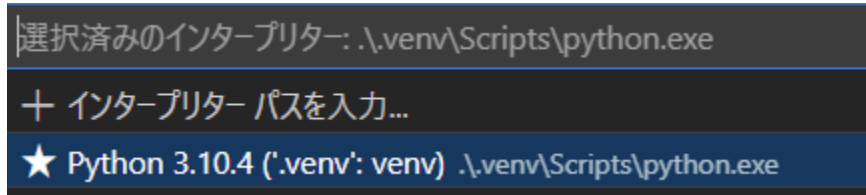
```
mkdir hello
cd hello
py -3 -m venv .venv
code .
```

仮想環境を構築すると、Python に関するコマンドなどが `.venv` フォルダにコピーされます。追加のパッケージは `.venv` フォルダにインストールして使用します。

1.4 VS Code で使用する Python インタプリターの選択

VS Code のプロジェクトで利用する Python インタプリターを選択します。

1. [表示]-[コマンドパレットを開く] を選択します。
2. [Python: インタプリタを開く] を選択します。
3. .venv の Python を選択します。

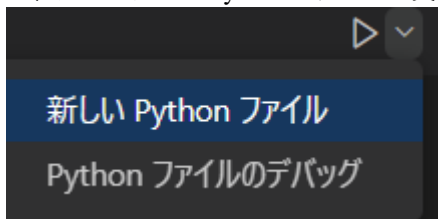


1.5 Python ファイルの作成と実行

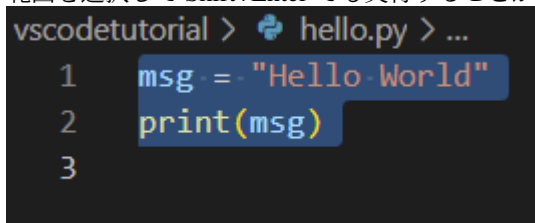
1. ワークスペース内に hello.py を作成します。
2. 次のソースコードを入力します。

```
msg = "Hello World"
print(msg)
```

3. VS Code 右上の Run ボタンの右側の下向きの印をクリックして [新しい Python ファイル] をクリックすると、ターミナルで Python ファイルが実行されます。



4. 範囲を選択して Shift+Enter でも実行することができます。

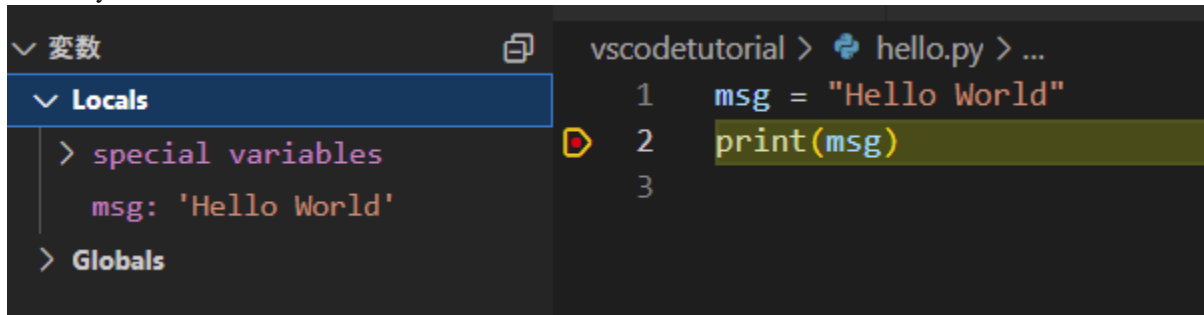


1.6 Python ファイルのデバッグ

2 行目にカーソルを移動し `m` す。F9 キーを押すとブレークポイントが設定されます。

```
vscode tutorial > hello.py > ...
1 msg = "Hello World"
2 print(msg)
3
```

VS Code 右上の、Run ボタンの右側の下向きの印をクリックします。[Python ファイルのデバッグ] をクリックすると、Python ファイルが実行され、ブレークポイントで停止します。

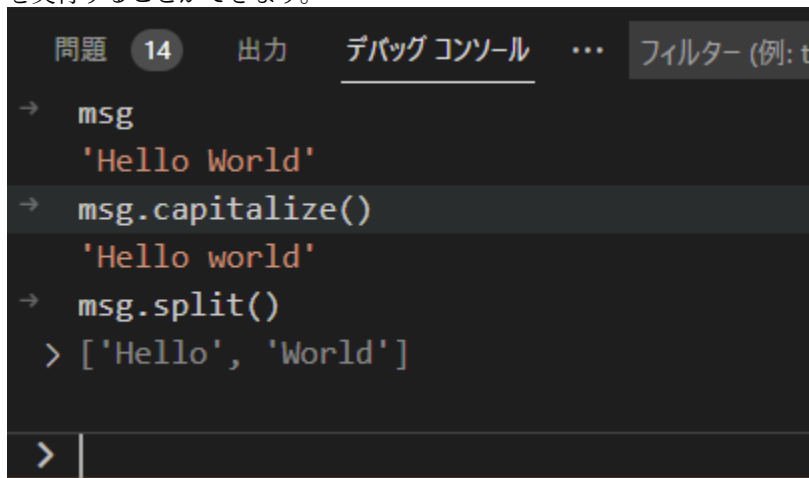


```
vscode tutorial > hello.py > ...
1 msg = "Hello World"
2 print(msg)
3
```

変数

- Locals
 - special variables
 - msg: 'Hello World'
 - Globals

画面下部の統合ターミナルの [デバッグ コンソール] で停止中の Python ファイルに対して、Python のプログラムを実行することができます。



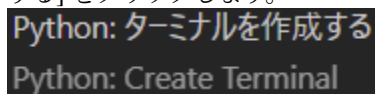
問題 14 出力 デバッグ コンソール ... フィルター (例: t

```
→ msg
'Hello World'
→ msg.capitalize()
'Hello world'
→ msg.split()
> ['Hello', 'World']
> |
```

1.7 パッケージの追加

Python で数値計算やグラフ描画に使用される `matplotlib` と `numpy` を使用します。

Python 仮想環境が有効な統合ターミナルで作業を行います。Ctrl+Shift+P を押し、[Python ターミナルを作成する] をクリックします。



```
Python: ターミナルを作成する
Python: Create Terminal
```

パッケージをインストールします。

```
pip install matplotlib
```

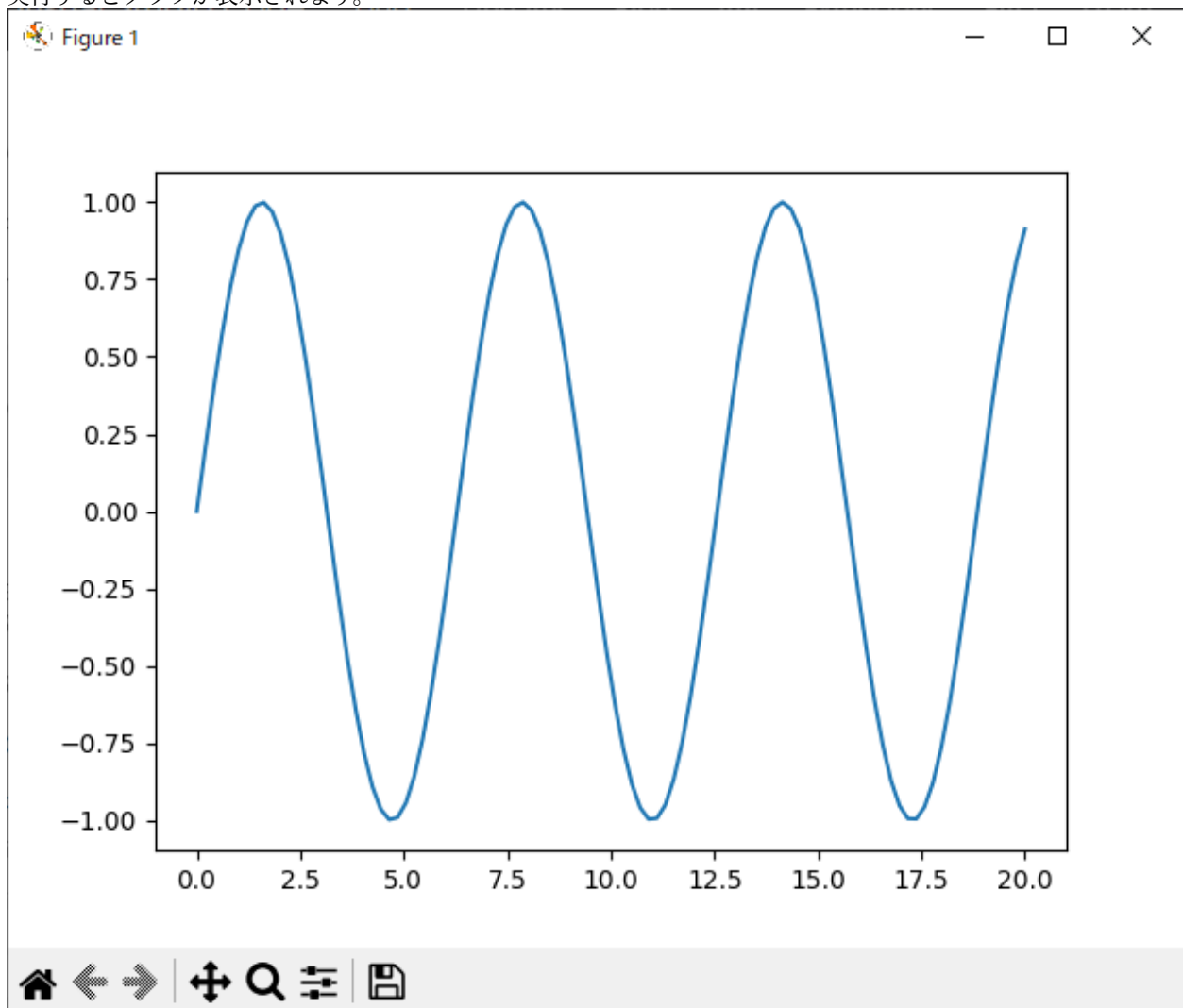
1.8 グラフ描画用 Python ファイルの作成

numpy と matplotlib を使用したプログラムを作成します。

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 20, 100)
plt.plot(x, np.sin(x))
plt.show()
```

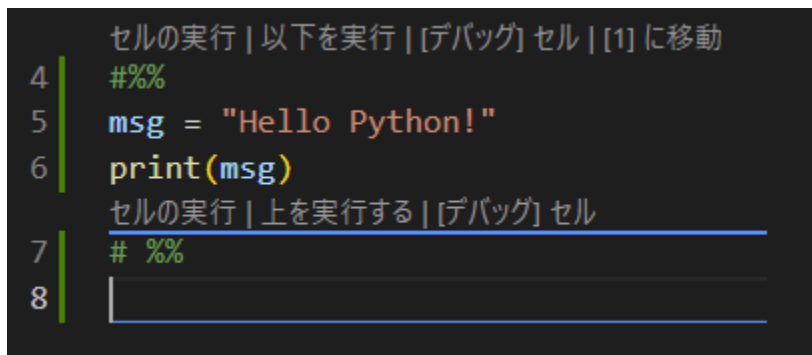
実行するとグラフが表示されます。



1.9 Jupyter Notebook の拡張機能の利用

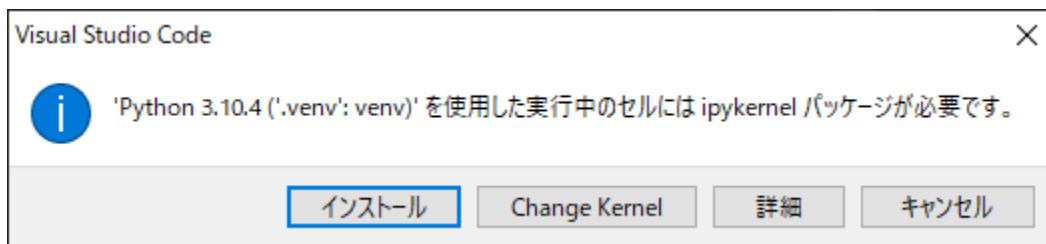
Python 拡張機能をインストールすると、Jupyter Notebook の拡張機能もインストールされます。

Python ファイルで`#%%` と入力すると、Jupyter Notebook の機能を使用することができます。

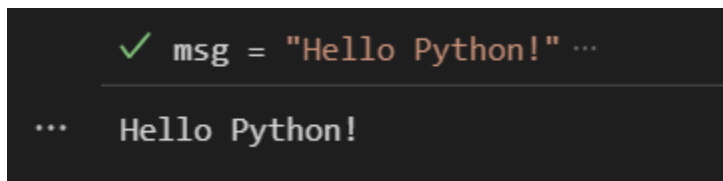


```
4 | セルの実行 | 以下を実行 | [デバッグ] セル | [1] に移動
5 | #%%
6 | msg = "Hello Python!"
7 | print(msg)
8 | セルの実行 | 上を実行する | [デバッグ] セル
   | # %%
```

最初だけ次のメッセージが表示されます。[インストール] をクリックします。



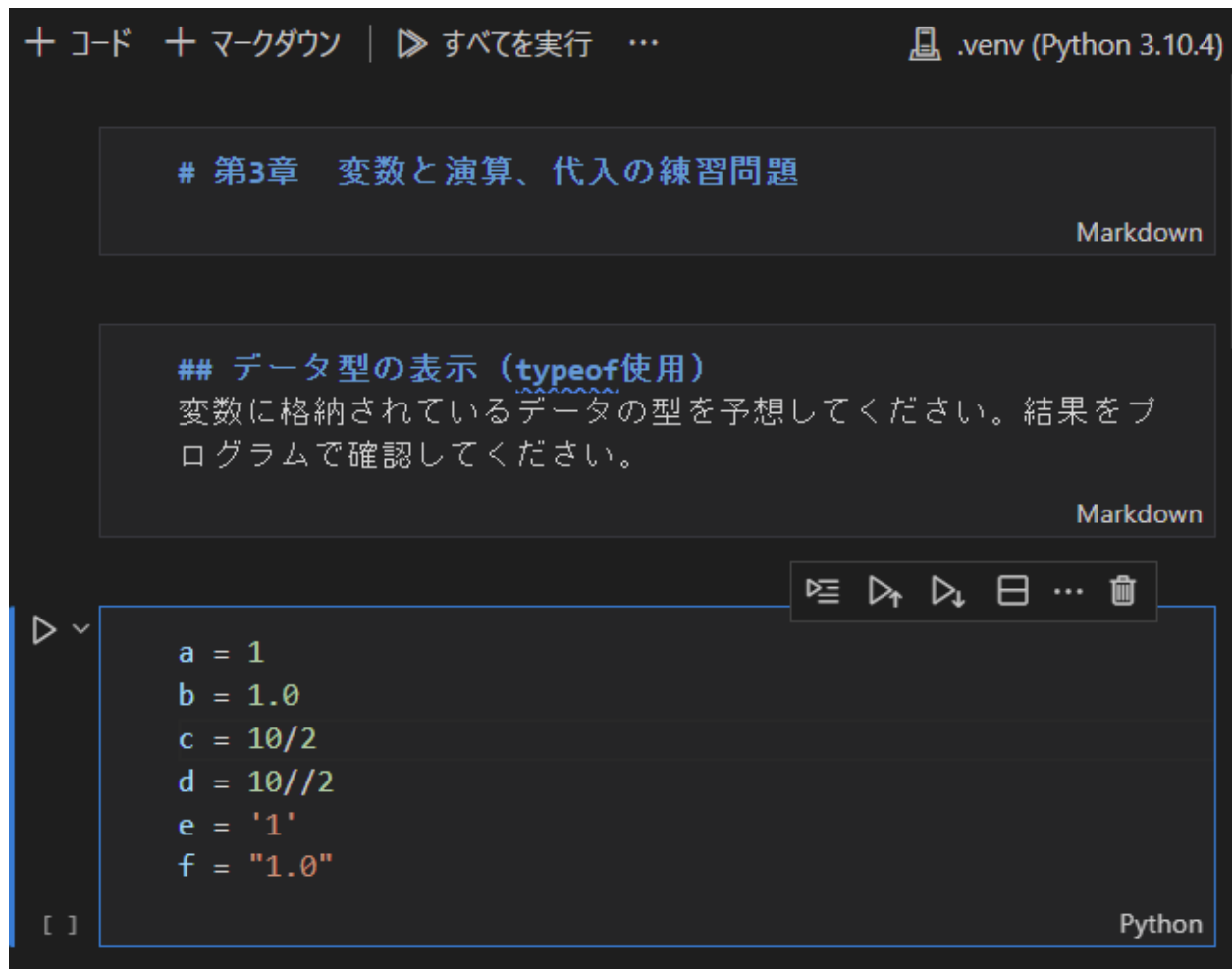
実行結果は右側の新しい画面に表示されます。



```
✓ msg = "Hello Python!" ...
... Hello Python!
```

デバッグもできます。

ファイル名の拡張子を`.ipynb`にすると、Jupyter Notebook を利用することができます。



説明、Python プログラムとその結果を、1つのファイルに保存することができるため便利です。

2. Visual Studio Code チュートリアル (Flask 基礎)

Python は大学工学部や企業の研究所などで、統計解析や人口知能（AI）の開発を中心に人気が出ました。その後、RPA(Robotic Process Automation)での採用や、入門者に優しい言語であること、**基本情報処理技術者資格**に採用されたことから、文系の大学や専門学校、高校などでも使用されるようになっていきます。このような人気の高まりを受けて、これまで専門的な言語で開発していた様々な情報システムを、Python を使用して開発ができるようになりつつあります。

Web システムもその 1 つです。現在、様々な取り組みが行われており、すでに実際のシステムで Python が使用されている例もあります。(モノタロウ [Tech Blog](#))

Python を使用して Web システムを開発するためには、一般的にフレームワークが用いられますが、現在でも、様々なフレームワークが提案・開発されています。代表的なフレームワークとして、Django、Bottle や Flask があり、今回は小中規模向けと言われている [Flask](#) を使用します。

Flask は"Micro Framework"と呼ばれ、Web システム開発に必要な多くの機能を直接サポートしていませんが、Flask 拡張として様々な Python パッケージを導入すると実現できます。

この資料は次のサイトの記事を参考に作成しています。

参照：[Flask Tutorial in Visual Studio Code](#)

2.1 チュートリアル用の環境構築

Python の仮想環境を利用します。

1. チュートリアル用のフォルダを作成します。その中に Python の仮想環境を作成します。

```
mkdir hello_flask
cd hello_flask
py -3 -m venv .venv
code .
```

2. VS Code に Python インタプリターを設定します。Ctrl+Shift+P でコマンドパレットを表示します。
Python: **インタプリターを選択**をクリックします。仮想環境のインタプリターを選択します。
3. Flask をインストールします。

```
pip install flask
```

ヒント: `py -3 -m pip install flask` と入力すると、Python のグローバル環境にインストールされます。仮想環境にインストールするときは `pip` コマンドを使用してください。

2.2 最小限の Flask アプリの作成

1. `hello_flask` に `app.py` を作成します。
2. `app.py` にプログラムを記述します。

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

@app.route("/")
def home():
    return "Hello, Flask!"
```

@で始まる行は**デコレータ**と呼ばれ、`home` 関数が実行される前に様々な処理が追加されています。例では、ブラウザを使用してサイトのルートにアクセスすると、`home` 関数が実行されます。そして、文字列 `Hello, Flask!` をブラウザに返します。その際、`HTML` などのコードは自動で追加されます。

3. アプリを起動します。

Python 仮想環境で起動する統合ターミナルを開きます。

注釈: `Ctrl+Shift+P` を押して、Python: **ターミナルを作成する**を選択します。

```
py -3 -m flask run
```

4. サイトのアクセスします。

統合ターミナル下部に表示される `http://127.0.0.1:5000/` を `Ctrl+Click` するとブラウザで開くことができます。

5. `Hello, Flask!` が表示されます。

注釈: ブラウザの `F12` キーを押すと、表示しているページのコードを確認することができます。

6. 統合ターミナルにアクセス状況が表示されます。

```
127.0.0.1 - - [24/Sep/2022 11:14:10] "GET / HTTP/1.1" 200 -
```

7. アプリを終了します。統合ターミナルで Ctrl+C で終了します。

2.3 アプリのデバッグ

Web アプリでも通常のアプリと同様にデバッグを行うことができます。

1. app.py を次のプログラムに修正します。

```
from flask import Flask
from datetime import datetime
import re # 正規表現ライブラリ

app = Flask(__name__)

@app.route("/")
def hoem():
    return "Hello, Flask!"

@app.route("/hello/<name>")
def hello_there(name):
    now = datetime.now()
    formatted_now = now.strftime("%A, %d %B, %Y at %X")

    # 変数 name で 1 文字以上のアルファベットで構成されている要素を抽出
    match_object = re.match("[a-zA-Z]+", name)

    if match_object: # 正規表現にマッチした情報が含まれているとき
        clean_name = match_object.group(0) # マッチした先頭要素を取得
    else: # 正規表現にした情報が含まれていないとき
        clean_name = "Friend"

    content = "Hello there, " + clean_name + "! It's " + formatted_now
    return content
```

2. ブレークポイントの設定 hello_there 関数の次の行 `now = datetime.now()` にブレークポイントを設定します。カーソルを移動して F9 キーを押すとブレークポイントを設定することができます。

Tip: ブレークポイントを設定したい行の、行番号の左側の空白をクリックしても、設定することができます。

3. デバッグ用の設定ファイルの作成 **実行とデバッグ** アイコンをクリックする (F5 キー) します。launch.json ファイルを作成するをクリックします。

2.3. アプリのデバッグ

launch.json に次の内容を記述します。

重要: 最初に起動される Python ファイルを"FLASK_APP"に記述します。VS Code のプロジェクトルート以外に app.py ファイルを配置する場合は変更が必要です。

例 プロジェクトルートに app.py を保存している

```
"FLASK_APP": "app.py"
```

例 プロジェクトルートの [vscode tutorial]-[hello_flask] フォルダ内に app.py ファイルを保存している

```
"FLASK_APP": "vscode tutorial.hello_flask.app"
```

```
{
    // IntelliSense を使用して利用可能な属性を学べます。
    // 既存の属性の説明をホバーして表示します。
    // 詳細情報は次を確認してください: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Python: Flask",
            "type": "python",
            "request": "launch",
            "module": "flask",
            "env": {
                "FLASK_APP": "vscode tutorial.hello_flask.app",
                "FLASK_DEBUG": "1"
            },
            "args": [
                "run",
                "--no-debugger",
                "--no-reload"
            ],
            "jinja": true,
            "justMyCode": true
        }
    ]
}
```

4. デバッグの開始

F5 キーを押すとデバッグを開始します。

5. ブラウザーでアクセス

デバッグを行うサイトにアクセスします。アドレス `http://127.0.0.1:5000/hello/VSCode` にアクセスすると、ブレークポイントで停止します。

2.4 テンプレートの利用

リターンする文字列で HTML を構築することもできますが、最近の Web アプリ開発はテンプレートエンジンと呼ばれる機能を使用することが一般的です。Flask は標準で Jinja2 テンプレートエンジンを使用することができます。

Flask では HTML テンプレートを `templates` フォルダに保存します。

1. `hello_there.html` を `templates` に作成します。
2. 次の内容を記述して保存します。Jinja2 は、`{{ }}` で囲まれた範囲が特別な意味を持ち、Python オブジェクトの参照やメソッドを使用することができます。`{% %}` は Jinja2 の機能で、条件分岐や繰り返しなどを行うことができます。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Hello, Flask</title>
  </head>
  <body>
    {%if name %}
      <strong>Hello there, {{ name }}!</strong> It's {{ date.strftime("%A,
←%d %B, %Y at %X") }}.
    {% else %}
      What's your name? Provide it after /hello/ in the URL.
    {% endif %}
  </body>
</html>
```

3. `app.py` を修正します。 `render_template` をインポートします。ブラウザからのアクセスで呼び出される関数の戻り値に、 `render_template` 関数を使用します。第 1 引数は `templates` に保存したファイル名、第 2 引数以降にテンプレートで使用するオブジェクトを渡します。

```
from flask import Flask
from datetime import datetime
from flask import render_template

app = Flask(__name__)

@app.route("/")
def home():
    return "Hello, Flask!"

@app.route("/hello/<name>")
def hello_there(name = None):
    return render_template(
        "hello_there.html", name=name, date=datetime.now())
```

(次のページに続く)

(前のページからの続き)

)

4. プログラムを実行して動作を確認してください。

2.5 static ファイルの使用

動的な Web サイトでは、HTML/CSS/JavaScript ファイルは**静的** (static) ファイルと呼ばれます。Flask は静的ファイルを static フォルダに保存します。

2.5.1 HTML ファイルの返却

1. static フォルダの作成 hello_flask に static フォルダを作成します。
2. CSS ファイルの作成 static フォルダに site.css ファイルを作成します。

```
.message {  
    font-weight: 600;  
    color: blue;  
}
```

3. HTML ファイルが CSS を参照するように変更します。templates/hello_there.html の<head> と</head>の間に次の内容を記述します。Flask は Web で使用するファイルの保存場所が決まっています。決まった場所に保存されているファイルを参照するときは url_for 関数を使用します。

```
<link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='site.  
↩css')}}" />
```

HTML にタグを追加します。次は修正済みの全内容です。

```
<!DOCTYPE html>  
<html lang="ja">  
<head>  
    <meta charset="UTF-8">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>Hello, Flask</title>  
  
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename=  
↩'site.css') }}" />  
</head>  
<body>
```

(次のページに続く)

(前のページからの続き)

```

    {%if name %}
        <span class="message">Hello there, {{ name }}!</span> It's {{ date.
↵strftime("%A, %d %B, %Y at %X") }}.
    {% else %}
        <span class="message">What's your name? Provide it after /hello/ in the
↵URL.</span>
    {% endif %}
</body>
</html>

```

4. 実行して CSS が反映されていることを確認します。

Tip: ブラウザの開発者ツール (F12) を使用すると、詳細を確認することができます。

2.5.2 json の返却

1. static フォルダに data.json ファイルを作成します。

```

{
  "01": {
    "note": "This data is very simple because we're demonstrating only the
↵mechanism."
  }
}

```

2. app.py に data.json を取得するためのルートを追加します。send_static_file 関数でファイルの内容をそのまま返却します。

```

@app.route("/api/data")
def get_data():
    return app.send_static_file("data.json")

```

3. http://127.0.0.0:5000/api/data にアクセスして、JSON データが取得できることを確認します。

2.6 ファイルの分割

これまでのサンプルは、Python プログラムを `app.py` に記述しました。プログラムのサイズが大きくなるとファイルを分割します。

1. `hello_app` ワークスペースを作成します。現在のフォルダ内に `hello_app` フォルダを作成します。
2. `hello_flask` フォルダから `static` と `templates` フォルダの内容を、`hello_app` フォルダにコピーします。
3. `hello_app` フォルダに `views.py` ファイルを作成します。

```
from flask import Flask
from flask import render_template
from datetime import datetime
from . import app

@app.route("/")
def home():
    return "Hello, Flask!"

@app.route("/hello/")
@app.route("/hello/<name>")
def hello_there(name = None):
    return render_template(
        "hello_there.html",
        name=name,
        date=datetime.now()
    )

@app.route("/api/data")
def get_data():
    return app.send_static_file("data.json")
```

4. `hello_app` フォルダに `__init__.py` ファイルを作成します。

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

5. `hello_app` フォルダに `webapp.py` ファイルを作成します。

```
from . import app
from . import views
```

6. `launch.json` ファイルを修正します。

```
"env": {
    "FLASK_APP": "hello_app.webapp",
```

(次のページに続く)

(前のページからの続き)

```
    "FLASK_DEBUG": "1"  
  },
```

7. プログラムを実行して動作を確認します。

2.7 積み残し

2.7.1 requirements.txt ファイル

作成したソースコードを別の環境で実行するためには、Python のバージョンや追加したパッケージを準備する必要があります。同じ環境を容易に作成するために requirements.txt ファイルを使用することができます。

2.7.2 コンテナ化

アプリケーションを本番環境に配置（デプロイ）する必要があります。デプロイを容易に行うために Docker を利用することができます。