


TOPICS Flask Tutorial ▼

## Visual Studio Code中的Flask教程

 (https://github.com/Microsoft/vscode-docs/blob/master/docs/python/tutorial-flask.md)

Flask是用于Web应用程序的轻量级Python框架，提供了URL路由和页面呈现的基础。

Flask之所以称为“微型”框架，是因为它不直接提供表单验证，数据库抽象，身份验证等功能。这些功能由称为Flask扩展的特殊Python包提供。这些扩展与Flask无缝集成，因此看起来好像它们是Flask本身的一部分。例如，Flask不提供页面模板引擎，但是默认情况下，安装Flask包括Jinja模板引擎。为了方便起见，我们通常将这些默认值称为Flask的一部分。

在此Flask教程中，您将创建一个简单的Flask应用程序，该应用程序包含三个使用通用基本模板的页面。在此过程中，您将体验Visual Studio Code的许多功能，包括使用终端，编辑器，调试器，代码段等。

该Flask教程的完整代码项目可以在GitHub上找到：[python-sample-vscode-flask-tutorial](https://github.com/Microsoft/python-sample-vscode-flask-tutorial) (https://github.com/Microsoft/python-sample-vscode-flask-tutorial)。

如果您有任何问题，请随时在VS Code文档库中提交 (https://github.com/Microsoft/vscode-docs/issues) 本教程的问题。

### 先决条件

要成功完成此Flask教程，您必须执行以下操作（与常规Python教程 (/docs/python/python-tutorial) 中的步骤相同）：

1. 安装Python扩展 (<https://marketplace.visualstudio.com/items?itemName=ms-python.python>)。
2. 安装Python 3版本（编写本教程）。选项包括：
  - （所有操作系统）从python.org (<https://www.python.org/downloads/>) 下载；通常使用页面上首先显示的Download Python 3.6.5按钮（或最新版本）。
  - （Linux）内置的Python 3安装效果很好，但是要安装其他Python软件包，您必须 `sudo apt install python3-pip` 在终端中运行。
  - （macOS）使用Homebrew (<https://brew.sh/>) 在macOS上进行 `brew install python3` 安装（不支持在macOS上进行Python的系统安装）。
  - （所有操作系统）从Anaconda (<https://www.anaconda.com/download/>) 下载（出于数据科学目的）。
3. 在Windows上，确保PATH环境变量中包含Python解释器的位置。您可以通过 `path` 在命令提示符处运行来检查位置。如果不包括Python解释器的文件夹，请打开Windows设置，搜索“环境”，选择“为您的帐户编辑环境变量”，然后编辑“路径”变量以包含该文件夹。

### 为Flask教程创建项目环境

在本部分中，您将创建一个安装Flask的虚拟环境。使用虚拟环境可以避免将Flask安装到全局Python环境中，并可以精确控制应用程序中使用的库。虚拟环境还使创建环境的requirements.txt文件变得容易。

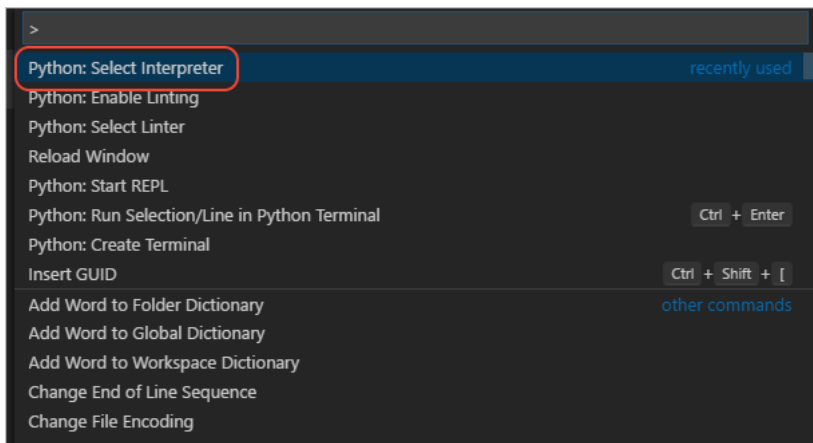
1. 在文件系统上，为本教程创建一个项目文件夹，例如 `hello_flask`。
2. 在该文件夹中，使用以下命令（适用于您的计算机）创建一个 `env` 基于当前解释器命名的虚拟环境：

```
# macOS/Linux
sudo apt-get install python3-venv    # If needed
python3 -m venv env

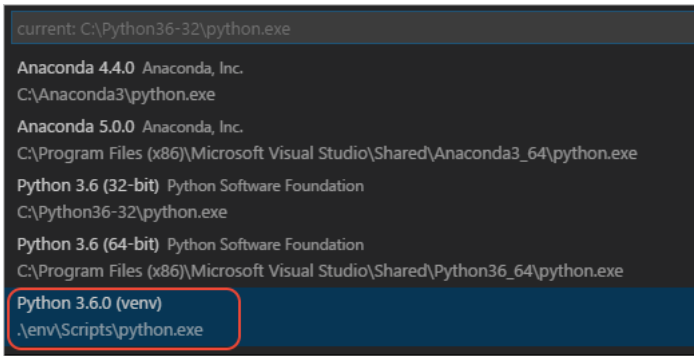
# Windows
python -m venv env
```

**注意：**运行上述命令时，请使用标准Python安装。如果您 `python.exe` 从Anaconda安装中使用，则会看到错误，因为`surepip`模块不可用，并且环境处于未完成状态。

3. 通过运行 `code .` 或通过运行VS Code并使用“文件” > “打开文件夹”命令来在VS Code中打开项目文件夹。
4. 在VS Code中，打开命令面板（“视图” > “命令面板”或（`Ctrl + Shift + P`））。然后选择Python：Select Interpreter命令：



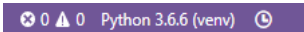
5. 该命令列出了VS Code可以自动定位的可用解释器的列表（您的列表会有所不同；如果看不到所需的解释器，请参阅配置Python环境 (/docs/python/environments)）。从列表中，在项目文件夹中选择以 `./env` 或开头的虚拟环境 `.\env`：



6. 运行终端：从命令面板**创建新的集成终端**（`Ctrl + Shift + ``），该命令面板将创建终端并通过运行其激活脚本自动激活虚拟环境。

**注意：**在Windows上，如果默认终端类型是PowerShell，则可能会看到一个错误，表明它无法运行`activate.ps1`，因为系统上已禁用了运行脚本。该错误提供了有关如何允许脚本的信息链接。否则，请使用“终端”：选择“默认Shell”将“命令提示符”或“Git Bash”设置为默认值。

7. 所选环境显示在VS Code状态栏的左侧，并注意“（venv）”指示器，该指示器告诉您正在使用虚拟环境：



8. 通过在VS Code终端中运行以下命令之一，在虚拟环境中安装Flask：

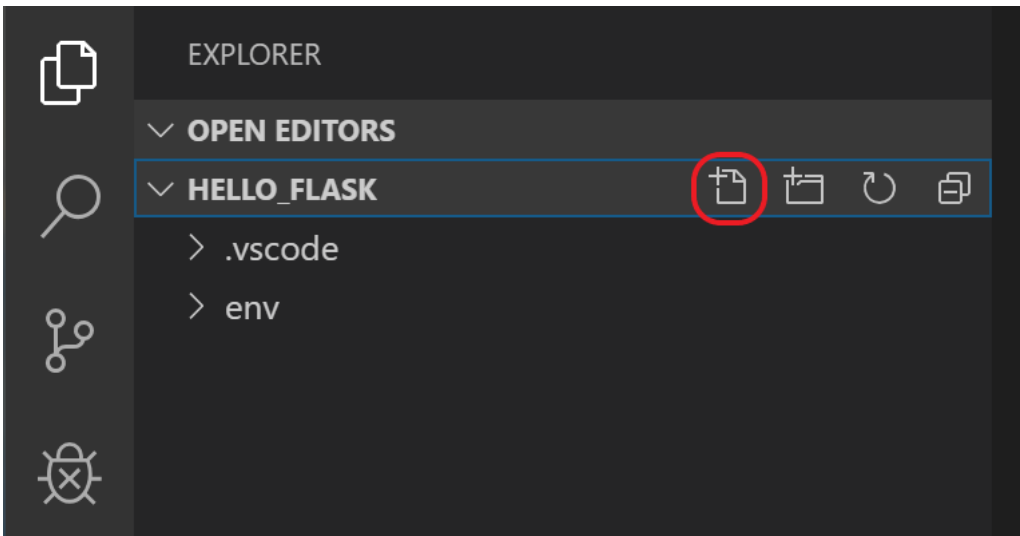
```
# macOS/Linux
pip3 install flask

# Windows
pip install flask
```

现在，您已经可以使用一个独立的环境来编写Flask代码。使用“终端：创建新的集成终端”时，“VS代码”会自动激活环境。如果打开单独的命令提示符或终端，请通过运行 `source env/bin/activate`（Linux / macOS）或 `env\scripts\activate`（Windows）激活环境。您知道当命令提示符开头显示（`env`）时环境已激活。

## 创建并运行最小的Flask应用

1. 在VS Code中，`app.py` 使用菜单中的File > New，按 `Ctrl + N`，或使用Explorer视图中的新文件图标（如下所示），在项目文件夹中创建一个新文件。



2. 在中 `app.py`，添加代码以导入Flask并创建Flask对象的实例。如果您在下面键入代码（而不是使用复制粘贴），则可以观察到VS Code的IntelliSense和自动完成功能（[docs/python/editing#\\_autocomplete-and-intellisense](https://code.visualstudio.com/docs/python/editing#_autocomplete-and-intellisense)）：

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

3. 同样在中 `app.py`，添加一个返回内容的函数（在这种情况下为简单字符串），并使用Flask的 `app.route` 装饰器将URL路由映射 / 到该函数：

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

**提示：**可以在同一功能上使用多个装饰器，每行一个，取决于要映射到同一功能的不同路由的数量。

4. 保存 `app.py` 文件（`Ctrl + S`）。

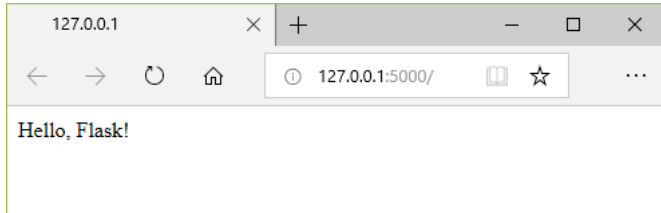
5. 在终端中，通过输入 `python3 -m flask run`（macOS / Linux）或 `python -m flask run`（Windows）（运行Flask开发服务器）来运行应用程序。`app.py` 默认情况下，开发服务器会查找。运行Flask时，应该看到类似于以下内容的输出：

```
(env) D:\py\\hello_flask>python -m flask run
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

如果看到无法找到Flask模块的错误，请确保已按照上一节末尾所述在虚拟环境中运行 `pip3 install flask` (macOS / Linux) 或 `pip install flask` (Windows)。

另外，如果要在其他IP地址或端口上运行开发服务器，请使用host和port命令行参数，与相同 `--host=0.0.0.0 --port=80`。

6. 要将默认浏览器打开到呈现的页面，请 `http://127.0.0.1:5000/` 在终端中 按住Ctrl键 并 单击 URL。



7. 观察到，当您访问类似/的URL时，调试终端中会显示一条消息，显示HTTP请求：

```
127.0.0.1 - - [11/Jul/2018 08:40:15] "GET / HTTP/1.1" 200 -
```

8. 通过在终端中使用 `Ctrl + C` 停止应用程序。

**提示：**如果要使用不同于的文件名 `app.py`，例如 `program.py`，请定义一个名为`FLASK_APP`的环境变量，并将其值设置为所选文件。然后Flask的开发服务器使用`FLASK_APP`的值而不是默认文件 `app.py`。有关更多信息，请参见Flask命令行界面 (<http://flask.pocoo.org/docs/1.0/cli/>)。

## 在调试器中运行应用

调试使您有机会在特定的代码行上暂停正在运行的程序。程序暂停后，您可以检查变量，在Debug Console面板中运行代码，或者利用Debugging中 (/docs/python/debugging)描述的功能。运行调试器还会在调试会话开始之前自动保存所有已修改的文件。

**开始之前：**通过在终端中使用 `Ctrl + C`，确保已在最后一节的结尾停止了正在运行的应用程序。如果您让应用程序在一个终端上运行，它将继续拥有该端口。结果，当您使用相同的端口在调试器中运行该应用程序时，原始正在运行的应用程序会处理所有请求，并且您将看不到正在调试的应用程序中的任何活动，并且该程序不会在断点处停止。换句话说，如果调试器似乎无法正常工作，请确保没有其他应用程序实例在运行。

1. 用 `app.py` 下面的代码替换的内容，这将添加第二条路由和函数，您可以在调试器中逐步进行操作：

```
from flask import Flask
from datetime import datetime
import re

app = Flask(__name__)

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

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

    # Filter the name argument to letters only using regular expressions. URL arguments
    # can contain arbitrary text, so we restrict to safe characters only.
    match_object = re.match("[a-zA-Z]+", name)

    if match_object:
        clean_name = match_object.group(0)
    else:
        clean_name = "Friend"

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

用于新URL路由的修饰符 `/hello/<name>` 定义了可以接受任何其他值的端点 `/hello/`。路由内部 `<` 和内部的标识符 `>` 定义了一个变量，该变量将传递给函数，并且可以在代码中使用。

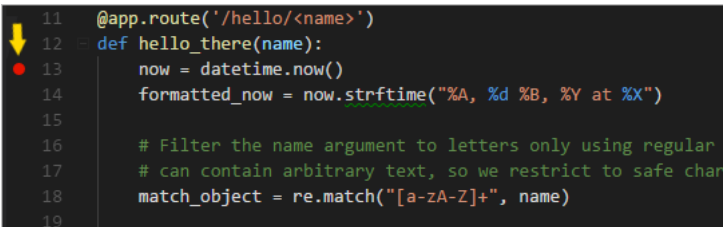
URL路由区分大小写。例如，路线 `/hello/<name>` 不同于 `/Hello/<name>`。如果要使用相同的函数来处理这两个函数，请对每个变体使用修饰符。

如代码注释中所述，请始终过滤用户提供的任意信息，以避免对您的应用程序进行各种攻击。在这种情况下，代码将`name`参数过滤为仅包含字母，从而避免了注入控制字符，HTML等。（当您在下一部分中使用模板时，Flask会自动过滤，因此您不需要此代码。）

2. 通过执行以下任一操作，在 `hello_there` 函数 (`now = datetime.now()`) 的第一行代码处设置一个断点：
  - 将光标放在该行上，按 `F9`，或，
  - 将光标放在该行上，选择“运行”>“切换断点”菜单命令，或者，

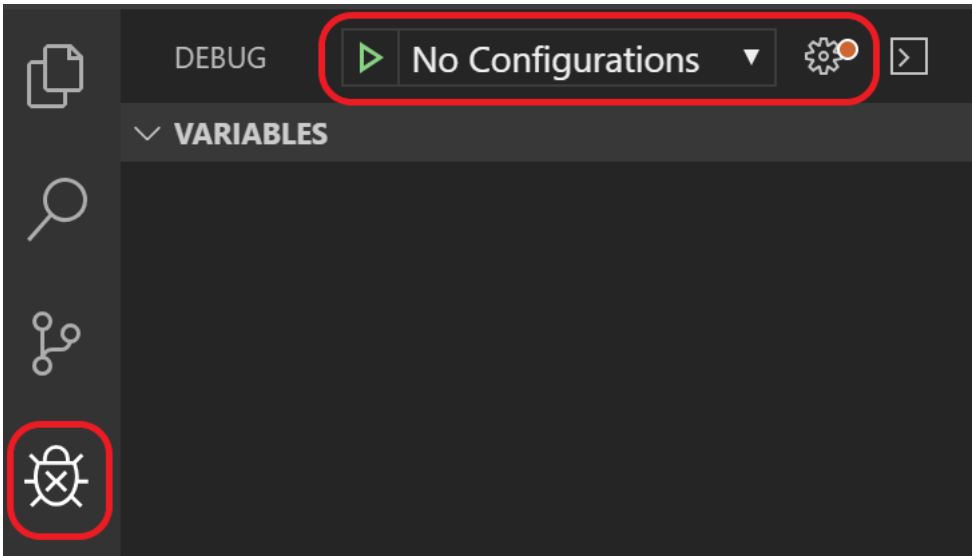
- 直接在行号左侧的空白处单击（将鼠标悬停在该行上时会出现一个淡红色的点）。

断点在左边距中显示为红点：



```
11 @app.route('/hello/<name>')
12 def hello_there(name):
13     now = datetime.now()
14     formatted_now = now.strftime("%A, %d %B, %Y at %X")
15
16     # Filter the name argument to letters only using regular
17     # can contain arbitrary text, so we restrict to safe char
18     match_object = re.match("[a-zA-Z]+", name)
19
```

3. 切换到VS Code中的“运行”视图（使用左侧活动栏）。在“运行”视图的顶部，齿轮图标上可能会显示“无配置”和警告点。这两个指示符都表示您还没有 `launch.json` 包含调试配置的文件：



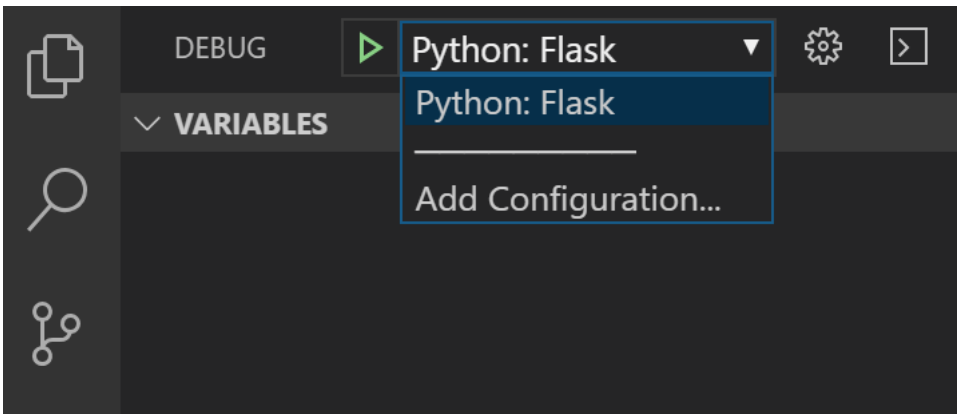
4. 选择齿轮图标，然后从出现的列表中选择**烧瓶**。VS Code创建并打开一个 `launch.json` 文件。此JSON文件包含调试配置，该配置是 `configuration` 数组中的JSON对象。
5. 向下滚动并检查配置，该配置名为“Python: Flask”。此配置包含 `"module": "flask"`，它告诉VS Code `-m flask` 在启动调试器时运行Python。它还在 `env` 属性中定义了FLASK\_APP环境变量，以标识启动文件，`app.py` 默认情况下，但是可让您轻松指定其他文件。如果要更改主机和/或端口，则可以使用 `args` 阵列。

```
{
  "name": "Python: Flask",
  "type": "python",
  "request": "launch",
  "module": "flask",
  "env": {
    "FLASK_APP": "app.py",
    "FLASK_ENV": "development",
    "FLASK_DEBUG": "0"
  },
  "args": [
    "run",
    "--no-debugger",
    "--no-reload"
  ],
  "jinja": true
},
```

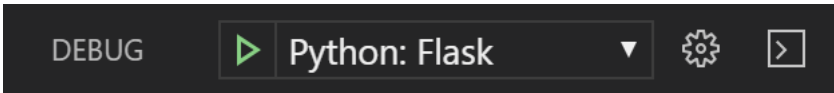
**注意：**如果 `env` 配置中的条目包含 `"FLASK_APP": "${workspaceFolder}/app.py"`，则将其更改为 `"FLASK_APP": "app.py"` 如上所示。否则，您可能会遇到诸如“无法导入模块C”之类的错误消息，其中C是项目文件夹所在的驱动器号。

**注意：**`launch.json` 创建后，“添加配置”按钮将出现在编辑器中。该按钮显示其他配置列表，以添加到配置列表的开头。（运行 > 添加配置菜单命令执行相同的操作。）。

6. 保存 `launch.json`（`Ctrl + S`）。在调试配置下拉列表（显示为Python: Current File）中，选择Python: Flask配置。



7. 通过选择“运行” > “开始调试”菜单命令，或选择列表旁边的绿色“开始调试”箭头（F5），启动调试器：



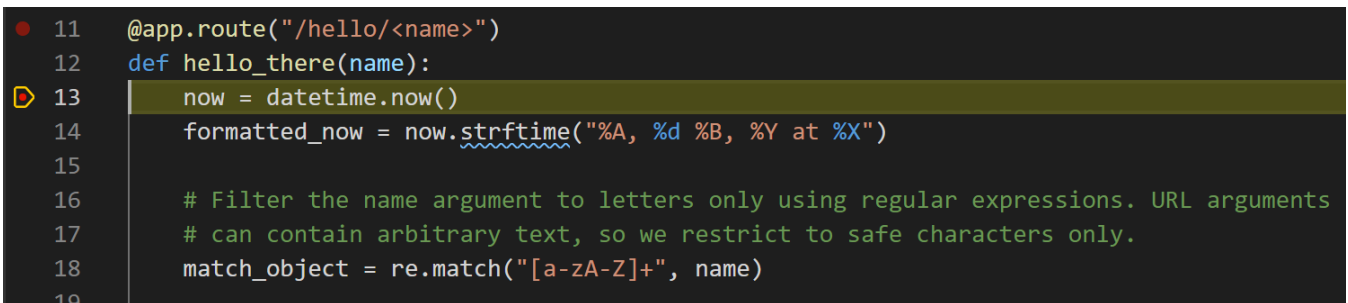
观察状态栏的颜色更改以指示调试：



VS Code中还将出现一个调试工具栏（如下所示），其中包含以下顺序的命令：暂停（或继续，F5），跨步（F10），单步进入（F11），单步退出（Shift + F11），重新启动（Ctrl + Shift + F5），然后停止（Shift + F5）。有关每个命令的说明，请参见VS代码调试 (/docs/editor/debugging)。

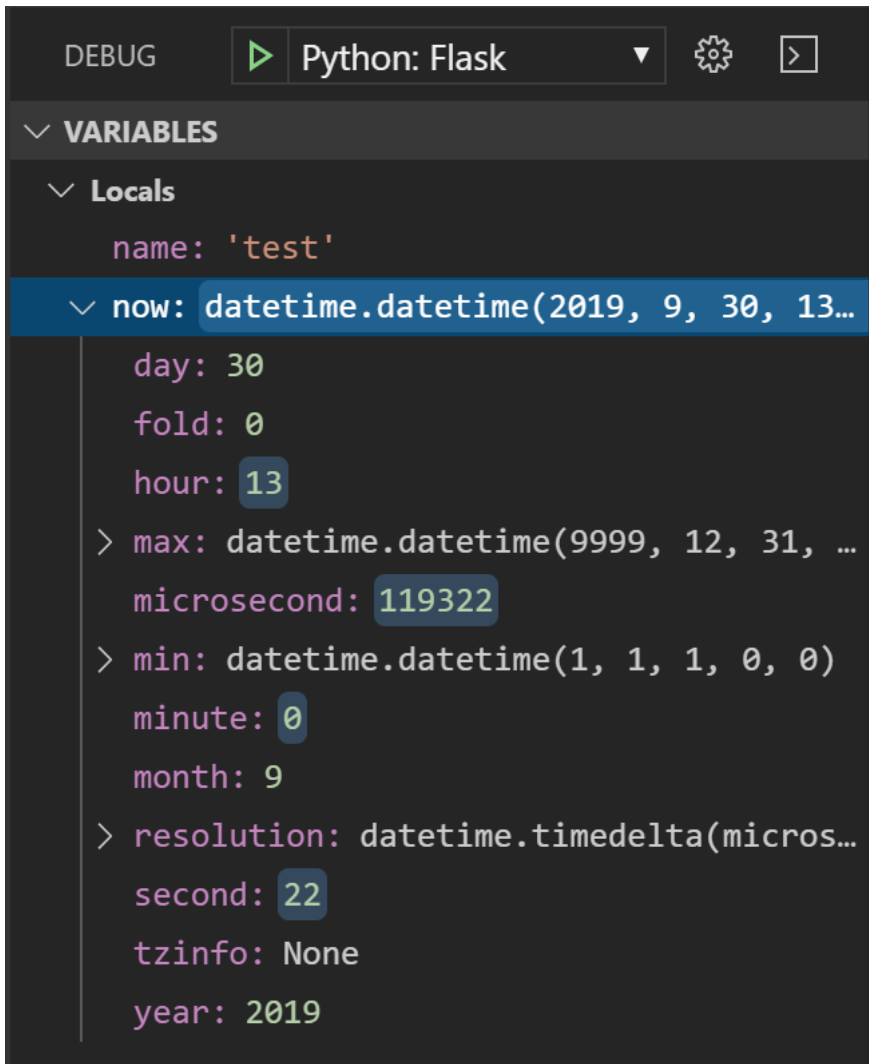


8. 输出显示在“Python调试控制台”终端中。Ctrl + 单击 该 <http://127.0.0.1:5000/> 终端中的链接以打开该URL的浏览器。在浏览器的地址栏中，导航到 <http://127.0.0.1:5000/hello/VSCode>。在页面呈现之前，VS Code在您设置的断点处暂停程序。断点上的黄色小箭头表示这是要运行的下一行代码。



9. 使用Step Over运行 `now = datetime.now()` 语句。

10. 在“VS代码”窗口的左侧，您会看到一个“变量”窗格，其中显示了局部变量（例如 `now`）以及参数（例如 `name`）。下面是Watch，Call Stack和Breakpoints的窗格（有关详细信息，请参见VS Code调试 (/docs/editor/debugging)）。在“本地”部分中，尝试扩展其他值。您也可以双击值（或使用 F2）来修改它们。但是，更改诸如 `now` 之类的变量可能会破坏程序。开发人员通常仅在代码没有产生正确的值时才进行更改以更正正确的值。



11. 程序暂停后，“**调试控制台**”面板（与“终端”面板中的“Python调试控制台”不同）使您可以试验表达式，并使用程序的当前状态尝试一些代码。例如，一旦您跨过了该行 `now = datetime.now()`，就可以尝试使用不同的日期/时间格式。在编辑器中，选择读取的代码 `now.strftime("%A, %d %B, %Y at %X")`，然后右键单击并选择 **Debug: Evaluate** 将该代码发送到运行它的调试控制台：

```
now.strftime("%A, %d %B, %Y at %X")
'Wednesday, 31 October, 2018 at 18:13:39'
```

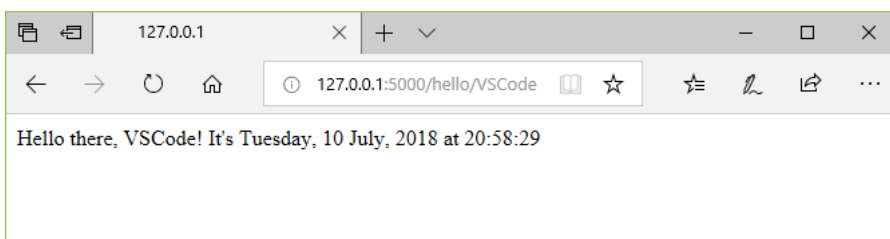
**提示：**调试控制台还显示了应用程序中可能未显示在终端中的异常。例如，如果在“运行”视图的“调用堆栈”区域中看到“在异常时暂停”消息，请切换到调试控制台以查看异常消息。

12. 将该行复制到调试控制台底部的>提示符，然后尝试更改格式：

```
now.strftime("%a, %d %B, %Y at %X")
'Wed, 31 October, 2018 at 18:13:39'
now.strftime("%a, %d %b, %Y at %X")
'Wed, 31 Oct, 2018 at 18:13:39'
now.strftime("%a, %d %b, %y at %X")
'Wed, 31 Oct, 18 at 18:13:39'
```

**注意：**如果看到自己喜欢的更改，则可以在调试会话期间将其复制并粘贴到编辑器中。但是，只有重新启动调试器后，这些更改才会应用。

13. 如果需要，可以再执行几行代码，然后选择继续（F5）以使程序运行。浏览器窗口显示结果：



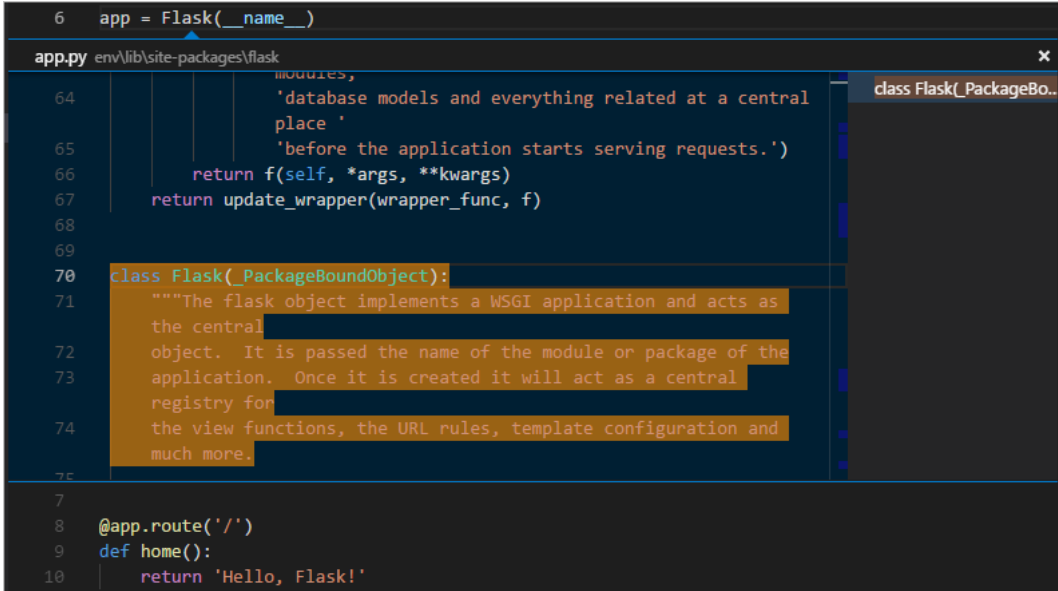
14. 完成后，关闭浏览器并停止调试器。要停止调试器，请使用“停止”工具栏按钮（红色方框）或“运行”>“**停止调试**”命令（Shift + F5）。

**提示：**为了更轻松地重复导航到特定的URL（例如） `http://127.0.0.1:5000/hello/VSCode`，请使用 `print` 语句输出该URL。该URL出现在终端中，您可以在其中使用 `Ctrl + click` 在浏览器中将其打开。

## 转到“定义”和“查看定义”命令

在使用Flask或任何其他库的过程中，您可能需要检查这些库本身中的代码。VS Code提供了两个方便的命令，这些命令可直接导航到任何代码中的类和其他对象的定义：

- **“转到定义”**从您的代码跳转到定义对象的代码。例如，在中 `app.py`，右键单击 `Flask` 类（在行中 `app = Flask(__name__)`），然后选择“**转到定义**”（或使用 `F12`），这将导航到Flask库中的类定义。
- **偷看定义**（`Alt + F12`，也在右键单击上下文菜单上）相似，但是直接在编辑器中显示类定义（在编辑器窗口中留出空间以避免混淆任何代码）。按 `Escape` 键关闭“窥视”窗口，或使用右上角的`x`。



## 使用模板渲染页面

到目前为止，您在本教程中创建的应用仅根据Python代码生成纯文本网页。尽管可以直接在代码中生成HTML，但开发人员避免了这种做法，因为它会使应用程序受到跨站点脚本（XSS）攻击（<http://flask.pocoo.org/docs/1.0/security/#cross-site-scripting-xss>）。`hello_there` 例如，在本教程的功能中，人们可能会考虑使用诸如之类的代码来格式化输出 `content = "<h1>Hello there, " + clean_name + "!</h1>`，其中将结果 `content` 直接提供给浏览器。此漏洞使攻击者可以在URL中放置恶意HTML（包括JavaScript代码），该URL最终 `clean_name` 在浏览器中运行。

更好的做法是使用**模板**将HTML完全排除在代码之外，这样您的代码仅与数据值有关，而与呈现无关。

- 模板是一个HTML文件，其中包含代码在运行时提供的值的占位符。模板引擎在呈现页面时会进行替换。因此，代码仅与数据值有关，而模板仅与标记有关。
- Flask的默认模板引擎是Jinja（<http://jinja.pocoo.org/>），在安装Flask时会自动安装。该引擎提供了灵活的选项，包括自动转义（以防止XSS攻击）和模板继承。通过继承，您可以定义具有通用标记的基础页面，然后在该基础上添加特定于页面的添加。

在本节中，您将使用模板创建一个页面。在以下各节中，您将应用程序配置为提供静态文件，然后为该应用程序创建多个页面，每个页面均包含基本模板中的导航栏。

1. 在该 `hello_flask` 文件夹内，创建一个名为的文件夹 `templates`，Flask默认在该文件夹中查找模板。
2. 在 `templates` 文件夹中，创建一个文件 `hello_there.html`，其内容如下。此模板包含两个名为“`name`”和“`date`”的占位符，分别由成对的花括号 `{{` 和分隔 `}}`。如您所见，您还可以在模板中直接包含格式代码：

```
<!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>
```

**提示：**Flask开发人员经常使用flask-babel（<https://pythonhosted.org/Flask-Babel/>）扩展名进行日期格式设置，而不是使用flask-babel（<https://pythonhosted.org/Flask-Babel/>）扩展名来 `strftime` 考虑区域设置和时区。

3. 在中 `app.py`，`render_template` 在文件顶部附近导入Flask的函数：

```
from flask import render_template
```



- 同样在中 `app.py`，修改该 `hello_there` 函数以用于 `render_template` 加载模板并应用命名的值（并添加路由以识别不带名称的大小写）。`render_template` 假定第一个参数是相对于 `templates` 文件夹的。通常，开发人员为模板命名的方式与使用它们的功能相同，但是不需要匹配的名称，因为您始终在代码中引用确切的文件名。

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

您可以看到代码现在变得更加简单，并且只关心数据值，因为标记和格式都包含在模板中。

- 启动程序（使用 `Ctrl + F5` 在调试器内部或外部），导航到 `/hello / name` URL，并观察结果。
- 也可以尝试使用一个名称导航到 `/hello / name` URL，例如 `<a%20value%20that%20could%20be%20HTML>`，查看Flask在工作中的自动转义。“名称”值在浏览器中显示为纯文本，而不是呈现实际元素。

## 服务静态文件

静态文件有两种类型。首先是页面模板可以直接引用的文件，例如样式表。此类文件可以位于应用程序的任何文件夹中，但通常放置在一个 `static` 文件夹中。

第二种类型是您要在代码中处理的类型，例如，当您要实现返回静态文件的API端点时。为此，Flask对象包含一个内置方法 `send_static_file`，该方法使用应用程序 `static` 文件夹中包含的静态文件生成响应。

以下各节演示了两种类型的静态文件。

### 引用模板中的静态文件

- 在 `hello_flask` 文件夹中，创建一个名为的文件夹 `static`。
- 在该 `static` 文件夹中，创建一个 `site.css` 包含以下内容的文件。输入此代码后，还请注意VS Code为CSS文件提供的语法突出显示，包括颜色预览：

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

- 在中 `templates/hello_there.html`，在 `</head>` 标记之前添加以下行，以创建对样式表的引用。

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

这里使用的Flask的`url_for` ([http://flask.pocoo.org/docs/0.12/api/#flask.url\\_for](http://flask.pocoo.org/docs/0.12/api/#flask.url_for))标记创建文件的适当路径。因为它可以接受变量作为参数，所以可以 `url_for` 根据需以编程方式控制生成的路径。

- 同样在中 `templates/hello_there.html`，用 `<body>` 以下使用 `message` 样式而不是 `<strong>` 标签的标记来替换 `content` 元素（如果您仅使用没有名称的 `hello / URL`，也会显示一条消息）：

```
{%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 %}
```

- 运行该应用程序，导航到 `/hello / name` URL，然后观察该消息呈蓝色显示。完成后，停止应用程序。

### 通过代码提供静态文件

- 在该 `static` 文件夹中，创建一个 `data.json` 具有以下内容的JSON数据文件（这些内容无意义的示例数据）：

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

- 在中 `app.py`，添加带有路由 `/api / data` 的函数，该函数使用以下 `send_static_file` 方法返回静态数据文件：

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

- 运行该应用程序，然后导航到 `/api / data` 端点以查看是否返回了静态文件。完成后，停止应用程序。

## 创建扩展基础模板的多个模板

因为大多数Web应用程序具有多个页面，并且由于这些页面通常共享许多公共元素，所以开发人员将这些公共元素分为一个基本页面模板，其他页面模板可以随后对其进行扩展。（这也称为模板继承。）



另外，由于您可能会创建许多扩展同一模板的页面，因此在VS Code中创建代码片段非常有用，您可以使用该代码片段快速初始化新的页面模板。摘要帮助您避免繁琐且容易出错的复制粘贴操作。

以下各节介绍了此过程的不同部分。

创建基础页面模板和样式

Flask中的基本页面模板包含一组页面的所有共享部分，包括对CSS文件，脚本文件等的引用。基本模板还定义一个或多个块标签，其他扩展基本模板的模板应被覆盖。块标记由基本模板和扩展模板界定，{% block <name> %} 并 {% endblock %} 在基本模板和扩展模板中都有。

以下步骤演示了如何创建基本模板。

- 1. 在 templates 文件夹中，创建一个 layout.html 包含以下内容的文件，其中包含名为“title”和“content”的块。如您所见，标记定义了一个简单的导航栏结构，该结构带有指向“主页”，“关于”和“联系人”页面的链接，您将在后面的部分中创建这些链接。每个链接再次使用Flask的 url\_for 标签在运行时为匹配的路由生成一个链接。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>{% block title %}{% endblock %}</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='site.css')}}" />
  </head>

  <body>
    <div class="navbar">
      <a href="{{ url_for('home') }}" class="navbar-brand">Home</a>
      <a href="{{ url_for('about') }}" class="navbar-item">About</a>
      <a href="{{ url_for('contact') }}" class="navbar-item">Contact</a>
    </div>

    <div class="body-content">
      {% block content %}
      {% endblock %}
      <hr/>
      <footer>
        <p>© 2018</p>
      </footer>
    </div>
  </body>
</html>
```

- 2. 将以下样式添加到 static/site.css 现有“消息”样式下方，然后保存文件。（本演练并不试图演示响应式设计；这些样式只会产生相当有趣的结果。）

```
.navbar {
  background-color: lightslategray;
  font-size: 1em;
  font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida Sans', Arial, sans-serif;
  color: white;
  padding: 8px 5px 8px 5px;
}

.navbar a {
  text-decoration: none;
  color: inherit;
}

.navbar-brand {
  font-size: 1.2em;
  font-weight: 600;
}

.navbar-item {
  font-variant: small-caps;
  margin-left: 30px;
}

.body-content {
  padding: 5px;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}
```

您现在可以运行该应用程序，但是由于您没有在任何地方使用基本模板，也没有更改任何代码文件，因此结果与上一步相同。完成其余部分以查看最终效果。

创建一个代码片段

由于您在下一节中创建的三个页面可以扩展 layout.html，因此可以节省创建代码片段的时间，该代码片段可以使用对基础模板的适当引用来初始化新模板文件。代码段可从单一来源提供一致的代码段，从而避免了在使用现有代码中的复制粘贴时可能蔓延的错误。

- 1. 在VS Code中，选择文件（Windows / Linux）或Code（macOS）菜单，然后选择首选项 > 用户片段。
- 2. 在出现的列表中，选择html。（如果您以前创建过代码段，则该选项在列表的“现有代码段”中可能显示为“html.json”。）
- 3. VS代码打开后 html.json，在现有花括号内添加以下条目（解释性注释（此处未显示）描述了详细信息，例如该 \$0 行如何指示VS代码在插入代码段后将光标放置在何处）：

```
"Flask Tutorial: template extending layout.html": {
  "prefix": "flexlayout",
  "body": [
    "{% extends \"layout.html\" %}",
    "{% block title %}",
    "$0",
    "{% endblock %}",
    "{% block content %}",
    "{% endblock %}"
  ],
  "description": "Boilerplate template that extends layout.html"
},
```

4. 保存 `html.json` 文件 ( `Ctrl + S` ) 。

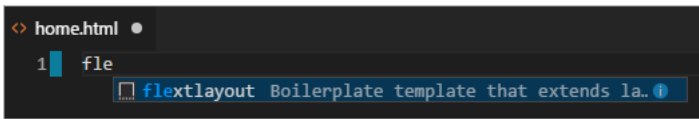
5. 现在，每当您开始输入代码段的前缀（例如）时 `flex`，VS Code 就会将代码段作为自动完成选项提供，如下一节所示。您也可以使用“**插入代码段**”命令从菜单中选择一个代码段。

有关一般代码段的更多信息，请参阅创建 (/docs/editor/userdefinedsnippets) 代码段 (/docs/editor/userdefinedsnippets)。

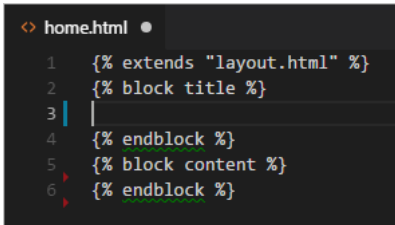
### 使用代码段添加页面

使用适当的代码段，您可以快速创建“主页”，“关于”和“联系人”页面的模板。

1. 在该 `templates` 文件夹中，创建一个名为的新文件 `home.html`，然后开始键入 `flex` 以查看该片段作为完成：



选择完成时，代码片段的代码将出现，并且光标位于代码片段的插入点上：



2. 在“标题”块的插入点，写入 `Home`，在“内容”块的插入点，写入 `<p>Home page for the Visual Studio Code Flask tutorial.</p>`，然后保存文件。这些行是扩展页面模板的唯一一部分：

3. 在 `templates` 文件夹中创建 `about.html`，使用代码片段插入样板标记，分别在 `About us` 和 `<p>About page for the Visual Studio Code Flask tutorial.</p>` 的“title”和“content”块中插入和，然后保存文件。

4. 重复上一步，`templates/contact.html` 使用 `Contact us` 和 `<p>Contact page for the Visual Studio Code Flask tutorial.</p>` 在两个内容块中创建。

5. 在中 `app.py`，为 `/about/` 和 `/contact/` 路由添加引用各自页面模板的功能。还要修改 `home` 功能以使用 `home.html` 模板。

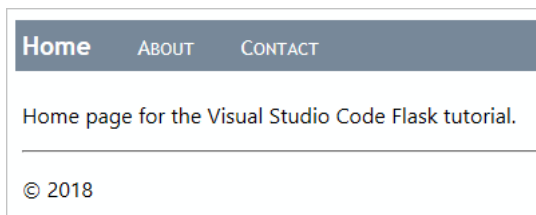
```
# Replace the existing home function with the one below
@app.route("/")
def home():
    return render_template("home.html")

# New functions
@app.route("/about/")
def about():
    return render_template("about.html")

@app.route("/contact/")
def contact():
    return render_template("contact.html")
```

### 运行应用

放置所有页面模板后，保存 `app.py`，运行应用程序并打开浏览器以查看结果。在页面之间导航以验证页面模板是否正确扩展了基础模板。



### 可选活动

以下各节介绍了在使用Python和Visual Studio Code时可能会有所帮助的其他步骤。

创建环境的requirements.txt文件

通过源代码控制或其他方式共享应用程序代码时，在虚拟环境中复制所有文件是没有意义的，因为收件人始终可以自己重新创建环境。

因此，开发人员通常会从源代码管理中忽略虚拟环境文件夹，而使用 requirements.txt 文件描述应用程序的依赖项。

尽管可以手动创建文件，但是也可以使用以下 pip freeze 命令根据已激活的环境中安装的确切库来生成文件：

1. 使用Python: Select Interpreter命令选择选定的环境后，运行Terminal: Create New Integrated Terminal命令（Ctrl + Shift + `）以打开一个激活了该环境的终端。
2. 在终端中，运行 pip freeze > requirements.txt 以 requirements.txt 在项目文件夹中创建文件。

收到项目副本的任何人（或任何构建服务器）都只需运行 pip install -r requirements.txt 命令以在原始环境中重新安装软件包。（但是，收件人仍然需要创建自己的虚拟环境。）

**注意：** pip freeze 列出当前环境中已安装的所有Python软件包，包括当前未使用的软件包。该命令还会列出具有确切版本号的软件包，您可能希望将其转换为范围，以在将来获得更大的灵活性。有关更多信息，请参阅pip命令文档中的需求文件 ([https://pip.readthedocs.io/en/stable/user\\_guide/#requirements-files](https://pip.readthedocs.io/en/stable/user_guide/#requirements-files))。

重构项目以支持进一步发展

在整个Flask教程中，所有应用程序代码都包含在一个 app.py 文件中。为了允许进一步开发并分离关注点，将这些部分重构 app.py 为单独的文件很有帮助。

1. 在您的项目文件夹中，为该应用程序创建一个文件夹，例如 hello\_app，以将其文件与其他项目级别文件（例如 requirements.txt 和 .vscode VS Code存储设置和调试配置文件的文件夹）分开。
2. 将 static 和 templates 文件夹移到中 hello\_app，因为这些文件夹中肯定包含应用程序代码。
3. 在该 hello\_app 文件夹中，创建一个名为的文件 views.py，其中包含路由和视图功能：

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

@app.route("/")
def home():
    return render_template("home.html")

@app.route("/about/")
def about():
    return render_template("about.html")

@app.route("/contact/")
def contact():
    return render_template("contact.html")

@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. 可选：右键单击编辑器，然后选择“排序导入”命令，该命令将合并来自相同模块的导入，删除未使用的导入，并对导入语句进行排序。使用上面代码中的命令，views.py 可以按以下方式更改导入（当然，您可以删除多余的行）：

```
from datetime import datetime

from flask import Flask, render_template

from . import app
```

5. 在 hello\_app 文件夹中，创建一个 \_\_init\_\_.py 包含以下内容的文件：

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

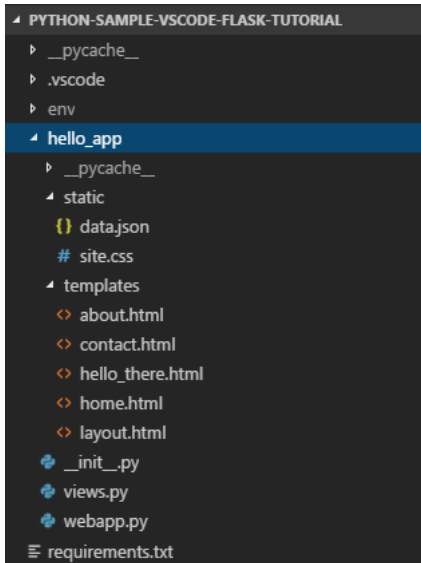
6. 在 hello\_app 文件夹中，创建一个 webapp.py 包含以下内容的文件：

```
# Entry point for the application.
from . import app # For application discovery by the 'flask' command.
from . import views # For import side-effects of setting up routes.
```

7. 打开调试配置文件 launch.json 并按 env 如下所示更新属性以指向启动对象：

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

- 删除 `app.py` 项目根目录中的原始文件，因为其内容已移至其他应用程序文件中。
- 您的项目的结构现在应类似于以下内容：



- 再次在调试器中运行该应用程序，以确保一切正常。要在VS Code调试器之外运行该应用程序，请在终端上执行以下步骤：
  - 为设置环境变量 `FLASK_APP`。在Linux和macOS上，使用 `export set FLASK_APP=webapp`；在Windows上使用 `set FLASK_APP=webapp`。
  - 导航到该 `hello_app` 文件夹，然后使用 `python3 -m flask run` (Linux / macOS) 或 `python -m flask run` (Windows) 启动程序。

#### 使用Docker扩展为Flask应用创建容器

该泊坞窗扩展 (<https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-docker>) 可以很容易地构建，管理和部署从Visual Studio代码集装箱的应用。如果您有兴趣学习如何为本教程中开发的Flask应用创建Python容器，请查看容器 (/docs/containers/quickstart-python) 教程中的Python (/docs/containers/quickstart-python)，它将逐步指导您：

- 创建一个 `Dockerfile` 描述简单Python容器的文件。
- 生成，运行和验证Flask (<http://flask.pocoo.org/>) 应用程序的功能。
- 调试在容器中运行的应用程序。

如果您有任何问题，请随时在VS Code文档库中提交 (<https://github.com/Microsoft/vscode-docs/issues>) 本教程的问题。

## 下一步

祝贺您完成在Visual Studio Code中使用Flask的演练！

可以在GitHub上找到本教程中完整的代码项目：`python-sample-vscode-flask-tutorial` (<https://github.com/Microsoft/python-sample-vscode-flask-tutorial>)。

由于本教程仅涉及页面模板的内容，因此请参阅Jinja2文档 (<http://jinja.pocoo.org/docs/>) 以获取有关模板的更多信息。该模板设计文档 (<http://jinja.pocoo.org/docs/templates/#synopsis>) 包含模板语言的所有细节。您可能还需要查看Flask官方教程 (<http://flask.pocoo.org/docs/1.0/tutorial/>) 以及Flask扩展 (<http://flask.pocoo.org/extensions/>) 的文档。

若要在生产网站上尝试您的应用程序，请查看教程使用Docker容器将Python应用程序部署到Azure App Service (<https://docs.microsoft.com/azure/python/tutorial-deploy-containers-01>)。Azure还提供了一个标准容器，即Linux上的App Service (<https://docs.microsoft.com/azure/python/tutorial-deploy-app-service-on-linux-01>)，您可以在VS Code中向其中部署Web应用程序。

您可能还想查看VS Code文档中与Python相关的以下文章：

- 编辑Python代码 (/docs/python/editing)
- 林亭 (/docs/python/linting)
- 管理Python环境 (/docs/python/environments)
- 调试Python (/docs/python/debugging)
- 测验 (/docs/python/testing)

如果您在本教程的过程中遇到任何问题，请随时在VS Code文档库中提出问题 (<https://github.com/Microsoft/vscode-docs/issues>)。

#### 该文档对您有帮助吗？

是      没有