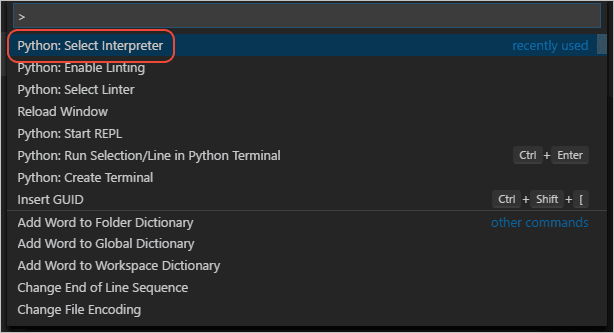
[Create a project environment for the Flask tutorial](https://code.visualstudio.com/docs/python/tutorial-flask" \l "_create-a-project-environment-for-the-flask-tutorial)

In this section, you create a virtual environment in which Flask is installed. Using a virtual environment avoids installing Flask into a global Python environment and gives you exact control over the libraries used in an application. A virtual environment also makes it easy to [Create a requirements.txt file for the environment](https://code.visualstudio.com/docs/python/tutorial-flask#_create-a-requirementstxt-file-for-the-environment).

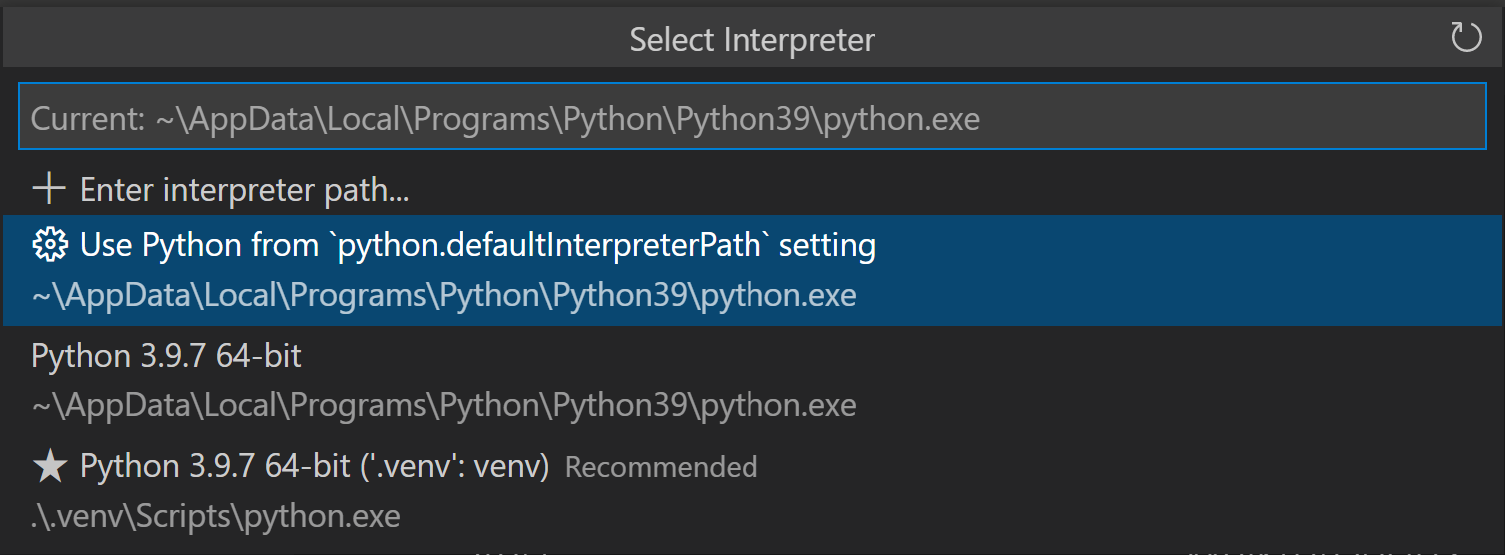
1. On your file system, create a project folder for this tutorial, such as hello\_flask.
2. In that folder, use the following command (as appropriate to your computer) to create and activate a virtual environment named .venv based on your current interpreter:
3. # Linux
4. sudo apt-get install python3-venv # If needed
5. python3 -m venv .venv
6. source .venv/bin/activate
7. # macOS
8. python3 -m venv .venv
9. source .venv/bin/activate
10. # Windows
11. py -3 -m venv .venv
12. .venv\scripts\activate

**Note**: Use a stock Python installation when running the above commands. If you use python.exe from an Anaconda installation, you see an error because the ensurepip module isn't available, and the environment is left in an unfinished state.

1. Open the project folder in VS Code by running code ., or by running VS Code and using the **File** > **Open Folder** command.
2. In VS Code, open the Command Palette (**View** > **Command Palette** or (Ctrl+Shift+P)). Then select the **Python: Select Interpreter** command:



1. The command presents a list of available interpreters that VS Code can locate automatically (your list will vary; if you don't see the desired interpreter, see [Configuring Python environments](https://code.visualstudio.com/docs/python/environments)). From the list, select the virtual environment in your project folder that starts with ./.venv or .\.venv:



1. Run [**Terminal: Create New Terminal**](https://code.visualstudio.com/docs/terminal/basics) (Ctrl+Shift+`)) from the Command Palette, which creates a terminal and automatically activates the virtual environment by running its activation script.

**Note**: On Windows, if your default terminal type is PowerShell, you may see an error that it cannot run activate.ps1 because running scripts is disabled on the system. The error provides a link for information on how to allow scripts. Otherwise, use **Terminal: Select Default Shell** to set "Command Prompt" or "Git Bash" as your default instead.

1. The selected environment appears on the right side of the VS Code status bar, and notice the **('.venv': venv)** indicator that tells you that you're using a virtual environment:

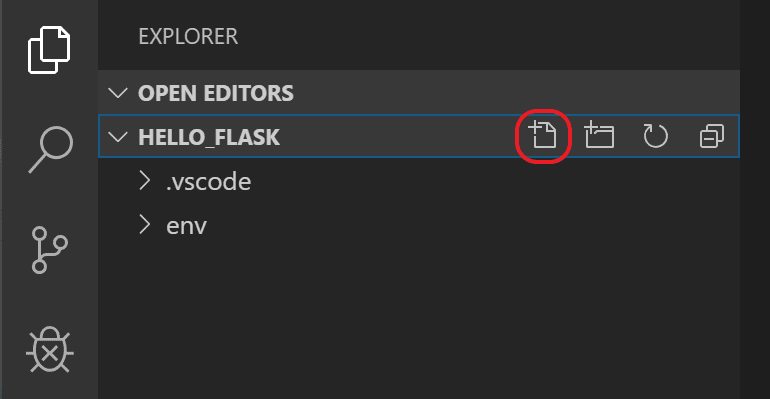
Flask tutorial: selected environment showing in the VS Code status bar

1. Update pip in the virtual environment by running the following command in the VS Code Terminal:
2. python -m pip install --upgrade pip
3. Install Flask in the virtual environment by running the following command in the VS Code Terminal:
4. python -m pip install flask

You now have a self-contained environment ready for writing Flask code. VS Code activates the environment automatically when you use **Terminal: Create New Terminal**. If you open a separate command prompt or terminal, activate the environment by running source .venv/bin/activate (Linux/macOS) or .venv\Scripts\Activate.ps1 (Windows). You know the environment is activated when the command prompt shows **(.venv)** at the beginning.

[Create and run a minimal Flask app](https://code.visualstudio.com/docs/python/tutorial-flask#_create-and-run-a-minimal-flask-app)

1. In VS Code, create a new file in your project folder named app.py using either **File** > **New** from the menu, pressing Ctrl+N, or using the new file icon in the Explorer View (shown below).



1. In app.py, add code to import Flask and create an instance of the Flask object. If you type the code below (instead of using copy-paste), you can observe VS Code's [IntelliSense and auto-completions](https://code.visualstudio.com/docs/python/editing#_autocomplete-and-intellisense):
2. from flask import Flask
3. app = Flask(\_\_name\_\_)
4. Also in app.py, add a function that returns content, in this case a simple string, and use Flask's app.route decorator to map the URL route / to that function:
5. @app.route("/")
6. def home():
7. return "Hello, Flask!"

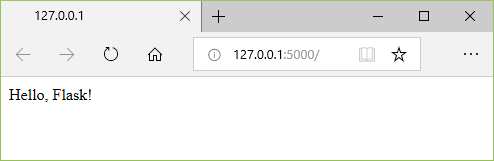
**Tip**: You can use multiple decorators on the same function, one per line, depending on how many different routes you want to map to the same function.

1. Save the app.py file (Ctrl+S).
2. In the Integrated Terminal, run the app by entering python -m flask run, which runs the Flask development server. The development server looks for app.py by default. When you run Flask, you should see output similar to the following:
3. (.venv) D:\py\\hello\_flask>python -m flask run
4. \* Environment: production
5. WARNING: Do not use the development server in a production environment.
6. Use a production WSGI server instead.
7. \* Debug mode: off
8. \* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

If you see an error that the Flask module cannot be found, make sure you've run python -m pip install flask in your virtual environment as described at the end of the previous section.

Also, if you want to run the development server on a different IP address or port, use the host and port command-line arguments, as with --host=0.0.0.0 --port=80.

1. To open your default browser to the rendered page, Ctrl+click the http://127.0.0.1:5000/ URL in the terminal.



1. Observe that when you visit a URL like /, a message appears in the debug terminal showing the HTTP request:
2. 127.0.0.1 - - [11/Jul/2018 08:40:15] "GET / HTTP/1.1" 200 -
3. Stop the app by using Ctrl+C in the terminal.

**Tip**: If you want to use a different filename than app.py, such as program.py, define an environment variable named FLASK\_APP and set its value to your chosen file. Flask's development server then uses the value of FLASK\_APP instead of the default file app.py. For more information, see [Flask command line interface](https://flask.palletsprojects.com/en/1.1.x/cli/).

[Run the app in the debugger](https://code.visualstudio.com/docs/python/tutorial-flask#_run-the-app-in-the-debugger)

Debugging gives you the opportunity to pause a running program on a particular line of code. When a program is paused, you can examine variables, run code in the Debug Console panel, and otherwise take advantage of the features described on [Debugging](https://code.visualstudio.com/docs/python/debugging). Running the debugger also automatically saves any modified files before the debugging session begins.

**Before you begin**: Make sure you've stopped the running app at the end of the last section by using Ctrl+C in the terminal. If you leave the app running in one terminal, it continues to own the port. As a result, when you run the app in the debugger using the same port, the original running app handles all the requests and you won't see any activity in the app being debugged and the program won't stop at breakpoints. In other words, if the debugger doesn't seem to be working, make sure that no other instance of the app is still running.

1. Replace the contents of app.py with the following code, which adds a second route and function that you can step through in the debugger:
2. from flask import Flask
3. from datetime import datetime
4. import re
5. app = Flask(\_\_name\_\_)
6. @app.route("/")
7. def home():
8. return "Hello, Flask!"
9. @app.route("/hello/<name>")
10. def hello\_there(name):
11. now = datetime.now()
12. formatted\_now = now.strftime("%A, %d %B, %Y at %X")
13. # Filter the name argument to letters only using regular expressions. URL arguments
14. # can contain arbitrary text, so we restrict to safe characters only.
15. match\_object = re.match("[a-zA-Z]+", name)
16. if match\_object:
17. clean\_name = match\_object.group(0)
18. else:
19. clean\_name = "Friend"
20. content = "Hello there, " + clean\_name + "! It's " + formatted\_now
21. return content

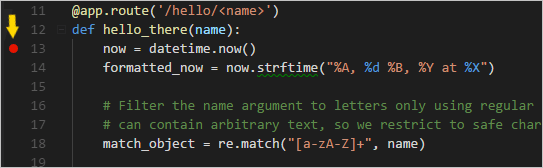
The decorator used for the new URL route, /hello/<name>, defines an endpoint /hello/ that can accept any additional value. The identifier inside < and > in the route defines a variable that is passed to the function and can be used in your code.

URL routes are case-sensitive. For example, the route /hello/<name> is distinct from /Hello/<name>. If you want the same function to handle both, use decorators for each variant.

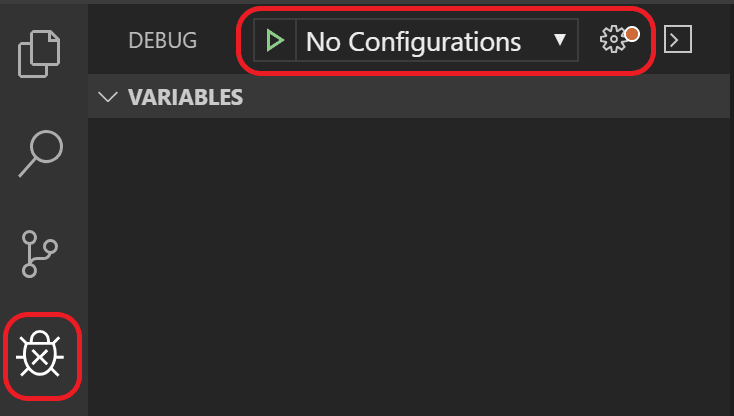
As described in the code comments, always filter arbitrary user-provided information to avoid various attacks on your app. In this case, the code filters the name argument to contain only letters, which avoids injection of control characters, HTML, and so forth. (When you use templates in the next section, Flask does automatic filtering and you won't need this code.)

1. Set a breakpoint at the first line of code in the hello\_there function (now = datetime.now()) by doing any one of the following:
   * With the cursor on that line, press F9, or,
   * With the cursor on that line, select the **Run** > **Toggle Breakpoint** menu command, or,
   * Click directly in the margin to the left of the line number (a faded red dot appears when hovering there).

The breakpoint appears as a red dot in the left margin:



1. Switch to **Run** view in VS Code (using the left-side activity bar or F5). You may see the message "To customize Run and Debug create a launch.json file". This means that you don't yet have a launch.json file containing debug configurations. VS Code can create that for you if you click on the **create a launch.json file** link:

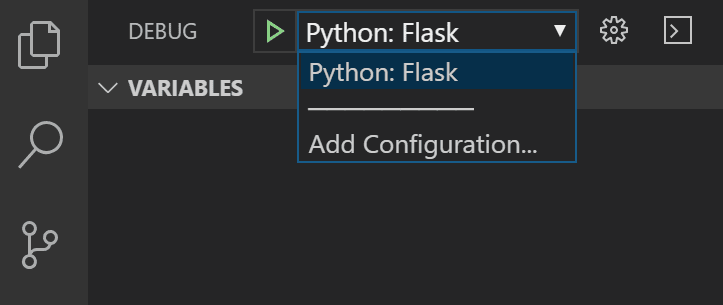


1. Select the link and VS Code will prompt for a debug configuration. Select **Flask** from the dropdown and VS Code will populate a new launch.json file with a Flask run configuration. The launch.json file contains a number of debugging configurations, each of which is a separate JSON object within the configuration array.
2. Scroll down to and examine the configuration, which is named "Python: Flask". This configuration contains "module": "flask",, which tells VS Code to run Python with -m flask when it starts the debugger. It also defines the FLASK\_APP environment variable in the env property to identify the startup file, which is app.py by default, but allows you to easily specify a different file. If you want to change the host and/or port, you can use the args array.
3. {
4. "name": "Python: Flask",
5. "type": "python",
6. "request": "launch",
7. "module": "flask",
8. "env": {
9. "FLASK\_APP": "app.py",
10. "FLASK\_ENV": "development",
11. "FLASK\_DEBUG": "0"
12. },
13. "args": [
14. "run",
15. "--no-debugger",
16. "--no-reload"
17. ],
18. "jinja": true
19. },

**Note**: If the env entry in your configuration contains "FLASK\_APP": "${workspaceFolder}/app.py", change it to "FLASK\_APP": "app.py" as shown above. Otherwise you may encounter error messages like "Cannot import module C" where C is the drive letter where your project folder resides.

**Note**: Once launch.json is created, an **Add Configuration** button appears in the editor. That button displays a list of additional configurations to add to the beginning of the configuration list. (The **Run** > **Add Configuration** menu command does the same action.).

1. Save launch.json (Ctrl+S). In the debug configuration dropdown list select the **Python: Flask** configuration.



1. Start the debugger by selecting the **Run** > **Start Debugging** menu command, or selecting the green **Start Debugging** arrow next to the list (F5):



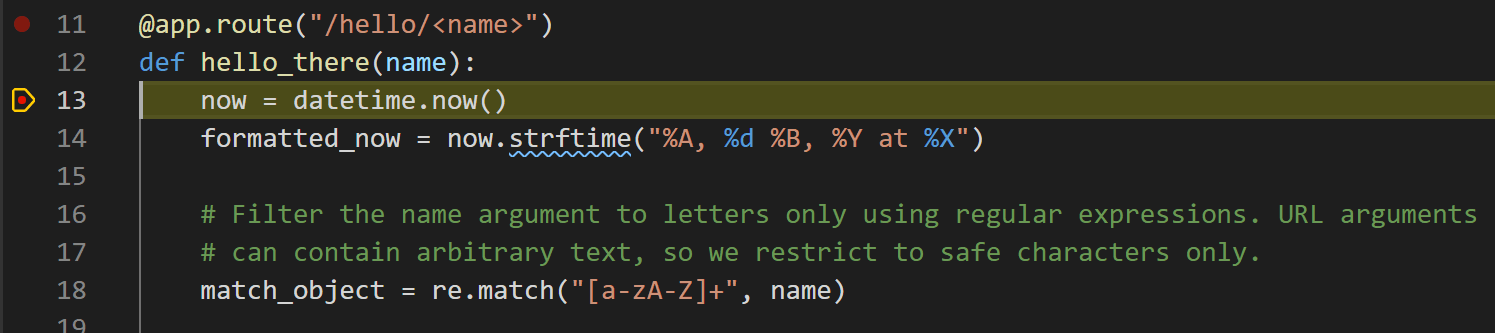
Observe that the status bar changes color to indicate debugging:

Flask tutorial: appearance of the debugging status bar

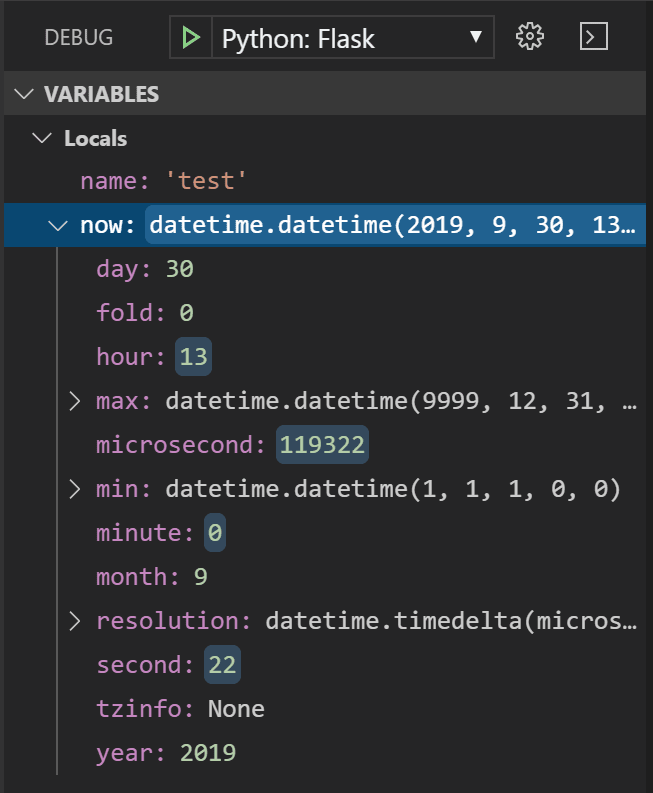
A debugging toolbar (shown below) also appears in VS Code containing commands in the following order: Pause (or Continue, F5), Step Over (F10), Step Into (F11), Step Out (Shift+F11), Restart (Ctrl+Shift+F5), and Stop (Shift+F5). See [VS Code debugging](https://code.visualstudio.com/docs/editor/debugging) for a description of each command.

Flask tutorial: the VS Code debug toolbar

1. Output appears in a "Python Debug Console" terminal. Ctrl+click the http://127.0.0.1:5000/ link in that terminal to open a browser to that URL. In the browser's address bar, navigate to http://127.0.0.1:5000/hello/VSCode. Before the page renders, VS Code pauses the program at the breakpoint you set. The small yellow arrow on the breakpoint indicates that it's the next line of code to run.



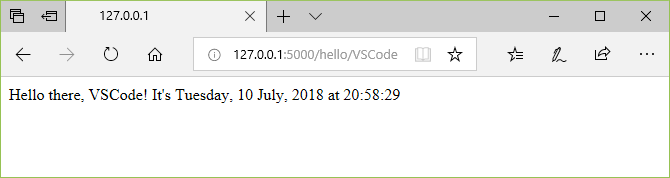
1. Use Step Over to run the now = datetime.now() statement.
2. On the left side of the VS Code window, you see a **Variables** pane that shows local variables, such as now, as well as arguments, such as name. Below that are panes for **Watch**, **Call Stack**, and **Breakpoints** (see [VS Code debugging](https://code.visualstudio.com/docs/editor/debugging) for details). In the **Locals** section, try expanding different values. You can also double-click values (or use F2) to modify them. Changing variables such as now, however, can break the program. Developers typically make changes only to correct values when the code didn't produce the right value to begin with.



1. When a program is paused, the **Debug Console** panel (which is different from the "Python Debug Console" in the Terminal panel) lets you experiment with expressions and try out bits of code using the current state of the program. For example, once you've stepped over the line now = datetime.now(), you might experiment with different date/time formats. In the editor, select the code that reads now.strftime("%A, %d %B, %Y at %X"), then right-click and select **Debug: Evaluate** to send that code to the debug console, where it runs:
2. now.strftime("%A, %d %B, %Y at %X")
3. 'Wednesday, 31 October, 2018 at 18:13:39'

**Tip**: The **Debug Console** also shows exceptions from within the app that may not appear in the terminal. For example, if you see a "Paused on exception" message in the **Call Stack** area of **Run and Debug** view, switch to the **Debug Console** to see the exception message.

1. Copy that line into the > prompt at the bottom of the debug console, and try changing the formatting:
2. now.strftime("%a, %d %B, %Y at %X")
3. 'Wed, 31 October, 2018 at 18:13:39'
4. now.strftime("%a, %d %b, %Y at %X")
5. 'Wed, 31 Oct, 2018 at 18:13:39'
6. now.strftime("%a, %d %b, %y at %X")
7. 'Wed, 31 Oct, 18 at 18:13:39'
8. Step through a few more lines of code, if you'd like, then select Continue (F5) to let the program run. The browser window shows the result:



1. Change the line in the code to use different datetime format, for example now.strftime("%a, %d %b, %y at %X"), and then save the file. The Flask server will automatically reload, which means the changes will be applied without the need to restart the debugger. Refresh the page on the browser to see the update.
2. Close the browser and stop the debugger when you're finished. To stop the debugger, use the Stop toolbar button (the red square) or the **Run** > **Stop Debugging** command (Shift+F5).

**Tip**: To make it easier to repeatedly navigate to a specific URL like http://127.0.0.1:5000/hello/VSCode, output that URL using a print statement. The URL appears in the terminal where you can use Ctrl+click to open it in a browser.

[Go to Definition and Peek Definition commands](https://code.visualstudio.com/docs/python/tutorial-flask#_go-to-definition-and-peek-definition-commands)

During your work with Flask or any other library, you may want to examine the code in those libraries themselves. VS Code provides two convenient commands that navigate directly to the definitions of classes and other objects in any code:

* **Go to Definition** jumps from your code into the code that defines an object. For example, in app.py, right-click on the Flask class (in the line app = Flask(\_\_name\_\_)) and select **Go to Definition** (or use F12), which navigates to the class definition in the Flask library.
* **Peek Definition** (Alt+F12, also on the right-click context menu), is similar, but displays the class definition directly in the editor (making space in the editor window to avoid obscuring any code). Press Escape to close the Peek window or use the **x** in the upper right corner.

