



# Developing Extensions for JupyterLab

# Meet the instructors



**Piyush Jain**

AWS

Piyush is a software engineer working on JupyterLab



**Alex Bozarth**

IBM

Alex is a software engineer working on Elyra and JupyterLab



**Martha Cryan**

IBM

Martha is a software engineer working on Elyra and JupyterLab

# Exploring Extensions

- What are extensions?
- Examples
  - DrawIO
  - Latex
  - Git
  - Spellchecker
  - Themes
- Installing prebuilt vs source



L<sup>A</sup>T<sub>E</sub>X



DrawIO

## Anatomy of an Extension

- Extensions, plugins and widgets
- Code Walkthrough...
  - We will be using an example created from <https://github.com/jupyterlab/extension-cookiecutter-ts>
  - The example is loosely based on the in-depth example found in the jupyterlab documentation: [https://jupyterlab.readthedocs.io/en/stable/extension/extension\\_tutorial.html](https://jupyterlab.readthedocs.io/en/stable/extension/extension_tutorial.html)

## Code Exercise

1. Open `examples/tutorial_extension` dir in your IDE
2. Try adding a toolbar button that refreshes the image using the following hints:
  - The toolbar can be accessed from `MainAreaWidget.toolbar`
  - A `ToolBarButton` class can be found in `@jupyterlab/apputils`
3. An example answer can be found on the next slide if you get stuck

## Code Exercise Example Answer

```
const button = new ToolbarButton({
  icon: refreshIcon,
  onClick: () => widget.load_image()
});
main.toolbar.addItem('refresh', button);
```

# Debugging JupyterLab Extensions

- When is debugging useful
- Setting up for debugging
- Launching JupyterLab for debugging
- Setting breakpoints
- Other ways to debug

# When is debugging useful

- To find errors in code
- Investigating unexpected results
- Understanding the code path
- Learning internals of other extensions

# How to know something has gone wrong

- UI elements are missing
- Errors appear in the server log
- Errors appear in the browser console

# Debugging in Visual Studio Code

- Instructions are in [DEBUGGING.md](#)
- Install the cookiecutter package  
`pip install cookiecutter`
- Use the debug-config-cookiecutter  
`cookiecutter ../debug-config-cookiecutter`
- Install the debug dependencies  
`jlpm install`
- Build the extensions  
`jlpm build`

# Setting breakpoints

The screenshot shows the JupyterLab interface with a Python code editor. The code is for a Jupyter server handler. A red dot on line 13 indicates a breakpoint is set at the `self.finish` call. The breadcrumb navigation shows the current location: `tutorial_extension > handlers.py > RouteHandler > get`.

```
tutorial_extension > handlers.py > RouteHandler > get
1  import json
2
3  from jupyter_server.base.handlers import APIHandler
4  from jupyter_server.utils import url_path_join
5  import tornado
6
7  class RouteHandler(APIHandler):
8      # The following decorator should be present on all verb methods (head, get, post,
9      # patch, put, delete, options) to ensure only authorized user can request the
10     # Jupyter server
11     @tornado.web.authenticated
12     def get(self):
13         self.finish(json.dumps({
14             "data": "This is /tutorial-extension/get_example endpoint!"
15         }))
16
17
18 def setup_handlers(web_app):
19     host_pattern = ".*$"
20
21     base_url = web_app.settings["base_url"]
22     route_pattern = url_path_join(base_url, "tutorial-extension", "get_example")
23     handlers = [(route_pattern, RouteHandler)]
24     web_app.add_handlers(host_pattern, handlers)
25
```

The sidebar on the left contains several panels:

- VARIABLES**: Currently empty.
- WATCH**: Currently empty.
- CALL STACK**: Currently empty.

At the top of the interface, there are tabs for `handlers.py`, `TS handler.ts`, `TS index.ts`, and `DEBUGGING.md U`. The `handlers.py` tab is active.

# Launching JupyterLab for debugging

The screenshot displays the JupyterLab interface with a Python file named `handlers.py` open. The code defines a `RouteHandler` class that inherits from `APIHandler`. The `get` method is decorated with `@tornado.web.authenticated` and returns a JSON response: `{ "data": "This is /tutorial-extension/get_example endpoint!" }`. A red dot on line 13 indicates a breakpoint.

The terminal output shows several HTTP requests and responses:

```
[W 2022-07-08 09:54:18.084 ServerApp] 404 GET /static/lab/3496.ecb0e7fcc54191234ae6.js.map?v=ec
0.0.1) 6.13ms referer=None
[W 2022-07-08 09:54:18.101 ServerApp] 404 GET /static/lab/4429.c4f083ef6b6e29345fd4.js.map?v=c4
0.0.1) 6.41ms referer=None
[W 2022-07-08 09:54:18.115 ServerApp] 404 GET /static/lab/4429.c4f083ef6b6e29345fd4.js.map?v=c4
0.0.1) 8.22ms referer=None
[W 2022-07-08 09:54:18.139 ServerApp] 404 GET /static/lab/7796.53c158c42e7f9697953b.js.map?v=53
0.0.1) 6.85ms referer=None
[W 2022-07-08 09:54:18.151 ServerApp] 404 GET /static/lab/7796.53c158c42e7f9697953b.js.map?v=53
0.0.1) 7.36ms referer=None
[W 2022-07-08 09:54:18.192 ServerApp] 404 GET /static/lab/714.d38baae8faccca175d4b.js.map?v=d38
0.0.1) 7.43ms referer=None
[W 2022-07-08 09:54:18.203 ServerApp] 404 GET /static/lab/714.d38baae8faccca175d4b.js.map?v=d38
0.0.1) 6.75ms referer=None
[I 2022-07-08 09:54:24.703 LabApp] Build is up to date
```

The left sidebar shows the following panels:

- WATCH**: Empty.
- CALL STACK**: Shows `jlab backend` (RUNNING) and `jlab frontend: localh...` (RUNNING).
- BREAKPOINTS**: Shows `handlers.py tutorial_extensi... 13` with a checked checkbox.

# Debugging Front End Extension

The image shows a VS Code editor window with the following components:

- Top Bar:** Shows the current file as `TS index.ts` and the active extension as `activate`.
- Left Sidebar:**
  - File Explorer:** Shows the project structure with `src > index.ts` selected.
  - CALL STACK:** Shows the call stack with `activate` in `src/index.ts` at line 48:5.
  - BREAKPOINTS:** Shows two breakpoints: one in `handlers.py` at `tutorial_extensi...` (line 13) and one in `index.ts` at `src` (line 48).
  - BROWSER BREAKPOINTS:** Shows no active browser breakpoints.
- Main Editor:** Displays the code for the `activate` function in `index.ts`. The line `app.commands.addCommand('tutorial:open', {` is highlighted. The code includes a `catch` block for missing extensions and a `execute` function that creates a `TutorialWidget` and adds it to the shell.
- Bottom Panel:** Shows the **TERMINAL** with server logs. The logs indicate successful GET requests for static files, such as `404 GET /static/lab/4429.c4f083ef6b6e29345fd4.js.map?v=c4`.

# Debugging Server Extension

The screenshot shows the JupyterLab IDE interface with the following components:

- File Explorer:** Shows the current file structure: `tutorial_extension > handlers.py > RouteHandler > get`.
- Code Editor:** Displays the `handlers.py` file with the following code:

```
1 import json
2
3 from jupyter_server.base.handlers import APIHandler
4 from jupyter_server.utils import url_path_join
5 import tornado
6
7 class RouteHandler(APIHandler):
8     # The following decorator should be present on all verb methods (head, get, post,
9     # patch, put, delete, options) to ensure only authorized user can request the
10    # Jupyter server
11    @tornado.web.authenticated
12    def get(self):
13        self.finish(json.dumps({
14            "data": "This is /tutorial-extension/get_example endpoint!"
15        })))
16
17
18 def setup_handlers(web_app):
19     host_pattern = ".*$"
```
- WATCH:** Empty panel.
- CALL STACK:** Shows the current execution path:
  - `get` in `handlers.py` at line 13:1 (PAUSED ON BREAKPOINT)
  - `wrapper` in `web.py` at line 3208:1
  - `_execute` in `web.py` at line 1711:1
  - `_run` in `events.py` at line 81:1
  - `_run_once` in `base_events.py`
  - `run_forever_base_event`
- BREAKPOINTS:** Shows two active breakpoints:
  - `handlers.py` `tutorial_extensi...` at line 13
  - `widaet.ts` `src` at line 8
- TERMINAL:** Displays the following log output:

```
.0.0.1) 7.39ms referer=None
[W 2022-07-08 10:02:08.469 ServerApp] 404 GET /static/lab/4429.c4f083ef6b6e29345fd4.js.map?v=c4
.0.0.1) 7.72ms referer=None
[W 2022-07-08 10:02:08.488 ServerApp] 404 GET /static/lab/7796.53c158c42e7f9697953b.js.map?v=53
.0.0.1) 7.98ms referer=None
[W 2022-07-08 10:02:08.502 ServerApp] 404 GET /static/lab/7796.53c158c42e7f9697953b.js.map?v=53
.0.0.1) 7.64ms referer=None
[W 2022-07-08 10:02:08.663 ServerApp] 404 GET /static/lab/jlab_core.fc632a0f38747f007842.js.map
(127.0.0.1) 5.43ms referer=None
[W 2022-07-08 10:02:08.674 ServerApp] 404 GET /static/lab/jlab_core.fc632a0f38747f007842.js.map
(127.0.0.1) 5.38ms referer=None
```

# Other ways to debug

- Front end extension
  - Use the browser directly to debug
- Server extensions
  - Python's command line debugger (pdb)

```
import pdb; pdb.set_trace()
```

- IPython pdb, a better alternative to pdb (pip install ipdb)

```
import ipdb; ipdb.set_trace()
```

## Working on Your Own Extension

- Jupyter Server extension
- Theme extension
- Whatever you wanted to start on - or pick from [here](#)