
JupyterLite Virtual Computing Environment (VCE) Software Guide (M348-24J)

The Open University

Nov 25, 2024

CONTENTS

I	Getting Started	3
1	Introduction	5
2	General comments and tips	7
2.1	Browser requirements	7
2.2	Choosing the OU hosted or the local VCE	7
2.3	Backing up your work	8
2.4	Updates and upgrades	8
2.5	Handling errors — don't panic	8
2.6	Where next?	9
II	Working with the Jupyter Environment	11
3	Using the VCE with JupyterLab	13
3.1	Launcher Buttons	14
4	Using Jupyter Notebooks	15
4.1	Managing notebooks	15
4.2	Notebook code kernels	15
4.3	Working with code cells	17
4.4	Working with Markdown cells	19
4.5	Coloured cell backgrounds	23
4.6	Navigating notebooks	24
III	Troubleshooting	27
5	Additional support	29
6	Troubleshooting	31
6.1	Accessing a terminal command-line interface within the VCE	31
6.2	Problems arising from working with large notebooks	33

6.3	Problems associated with running out of memory	33
6.4	Problems running notebook code	35
6.5	Finding version numbers for software in your VCE	35

IV Productivity 37

7 Jupyter Notebook Accessibility 39

7.1	Keyboard shortcuts	39
7.2	Visual Display Settings	42
7.3	Audible Alerts	43

8 File Management 47

8.1	Uploading and Downloading Files	47
8.2	Zippping and unzipping compressed archive files	47
8.3	Advanced file management	49

VCE Cribsheet

Accessing the hosted VCE:

- Access the hosted VCE via the **TM351 remote VCE** link in the `Resources` tab of the TM351 VLE;
- [hosted VCE documentation](https://docs.ocl.open.ac.uk/container-launcher/user/) — `https://docs.ocl.open.ac.uk/container-launcher/user/`

Important settings:

- Jupyter notebook server password / access token: `M348-24J`
- Home directory path inside VCE container: `/home/ou/M348-24J`

Docker image for local running: `ousefulcoursecontainers/ou-m348:24j`

Part I

Getting Started

INTRODUCTION

In this module, you will use a module specific virtual computing environment (VCE) for some of the module activities. The VCE is provided in various alternative forms, which offer an equivalent computational environment:

- a hosted environment using the Open Computing Lab remote hosting service;
- accessed via the module VLE as a JupyterLite environment running in a browser based HTML5 application
- a locally run environment distributed as a Docker container identical to the container used by the Open Computing Lab
- a locally run application that deploys the same JupyterLite environment as the VLE delivered HTML application.

The VCE provides a customised computing environment appropriate for your module and an interactive browser based user interface to access the software running inside it. The computational environments provided by each route provide equivalent Jupyter environments and execute the notebooks in the same way.

GENERAL COMMENTS AND TIPS

In this section you will find some general comments and tips associated with using the VCE.

2.1 Browser requirements

You will access the software tools provided by the virtual computing environment (either the hosted VCE or the local VCE) via a web browser. For VCEs running Jupyter environments, the notebooks have been tested extensively with the Chrome web browser. Recent versions of the Firefox, Edge and Safari web browsers should also work; Internet Explorer and older, non-Chromium versions of Edge are *not* supported and notebooks may not work correctly if you use them.

To make it easier to access a locally running VCE, we suggest that you add a browser bookmark for the default web page published by the locally running VCE container. For the remotely hosted VCE, we suggest that you access OpenComputing Lab via a bookmark for the module Resources page on the VLE.

2.2 Choosing the OU hosted or the local VCE

For many students, OpenComputing Lab provides the most convenient way of accessing the M348 VCE. All you need to access the hosted VCE is an internet connection and a computer running a modern web browser. For the duration of the module, all your work will be stored online in a personal file storage area. However, at the end of the module, you will lose access to the hosted VCE.

If you need to access the VCE in an offline environment, or if you prefer not to use the hosted environment, you can run the VCE locally on your own computer. The local environment also provides a way of using the VCE when the module has finished and the online VCE is no longer available.

The hosted VCE and the local VCE provide the same working environment, so your decision as to which environment to use is your own personal preference. Indeed you may want to make use of both environments, accessing each of them at different times or in different circumstances. However, when doing so, you will have to manage how you synchronise your files across the two VCEs yourself.

2.3 Backing up your work

It is generally regarded as good practice to make backup copies of any files that you would not like to lose. This applies to the contents of the shared folder that is established when setting up the local VCE, as well as all the folders within the hosted VCE.

In the local VCE, only the files you save inside the VCE directory that the shared folder has been mounted against (recommended as `/home/ou/M348-24J`) will be saved to the shared folder on your desktop. All files inside the VCE will persist inside the container until the container is deleted or destroyed.

Backups are regularly made of your files on the hosted VCE, although these will only be recovered in the event of hardware failure. For the local VCE, files will persist in the directory on your host computer that is mounted into the VCE, even if the VCE container is destroyed. In each case, you should maintain your own backups in case you accidentally delete, change or overwrite a file or its contents. This is especially important whilst you are working on your assessments where you may try a variety of approaches to answer the questions.

2.4 Updates and upgrades

In putting together the VCE, we have tried to ensure that all the software packages and their inter-connections run smoothly. Some of the configuration is software version specific, so you are strongly encouraged not to update or upgrade any of the software packages installed within the environment unless instructed to do so by the module team. Software updates and upgrades occasionally introduce changes that result in undesirable software behaviour, sometimes known as ‘breaking changes’, that cannot always be predicted in advance.

Information regarding any critical updates or changes recommended by the module team will be distributed via the module forums.

2.5 Handling errors — don’t panic

Hopefully, you should not see any error messages when installing or running the software provided by the VCE. On completion of each step everything should be working.

You should try to make sure you have the virtual computing environment up and running as soon as you are advised to access it in the module materials or study calendar. If there are any problems then there should be plenty of time to solve them.

If something doesn’t appear to be working, try to read through the error messages and see if you can tell what didn’t load properly. Most importantly of all, **don’t panic**.

Please remember when raising software issues that posting error logs can really help others to try to solve the problem. Saying ‘My software is broken/doesn’t work/prints out scary red or purple messages’ is not helpful. However, sharing those scary messages probably is. Most importantly of all, **don’t be embarrassed** about sharing error messages: they often contain the key to the solution of whatever problem caused them.

If you do encounter a problem, the section [Additional support](#) describes a general strategy for working through the problem and how to ask for help. A later section, [Troubleshooting](#), provides more specific guidance for working through particular sorts of issues with the different software applications.

2.6 Where next?

To get started with the online hosted VCE:

- access the hosted VCE via the **TM351 remote VCE** link in the `Resources` tab of the TM351 VLE;
- refer to the [hosted VCE documentation](#) for guidance on how to use the hosted VCE.

If you are new to working with Jupyter notebooks and/or JuoyterLab, check the sections on working with the JupyterLab environment ([Section 3](#)) and with Jupyter notebook document ([Section 4](#)) .

Experienced JupyterLab users may also find it useful to refer to the JupyterLab section for information on custom extensions installed in the M348 VCE.

Part II

Working with the Jupyter Environment

USING THE VCE WITH JUPYTERLAB

When the VCE is fully initialised, you will be presented with the JupyterLab browser based integrated development environment. JupyterLab allows you to manage files, run Jupyter notebooks, and access a terminal from within the same environment window, as depicted in [Figure 3.1](#).

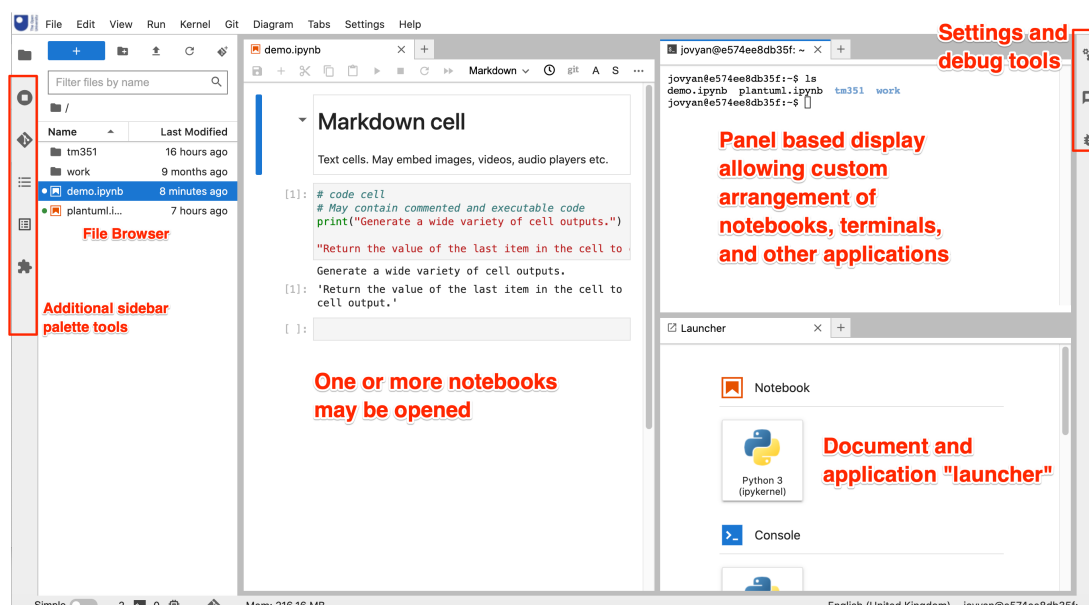


Figure 3.1: The JupyterLab user interface

A screenshot of the JupyterLab interactive development environment (IDE) style user interface, showing a file browser sidebar, document and application launcher, opened document and terminal editors and a menu toolbar.

JupyterLab provides an integrated development environment (IDE) that provides access to a file browser and a range of editors, including a fully featured Jupyter notebook editor.

The JupyterLab environment distributed as part of the VCE includes several pre-installed JupyterLab extensions that enhance the usability of the environment to support your studies (([Section 7 Jupyter Notebook Accessibility](#))).

Note: The JupyterLab extensions that have been preinstalled into the M348 VCE may differ from

extensions used in other module VCEs.

Some modules may distribute a “JupyterLab configuration pack” as a separately installable Python package. Such packages can be used to extend your own JupyterLab environment outside of the VCE in a way that matches the customisation of the environment within the VCE.

For example, the JupyterLab environment in the TM351 VCE is extended using the Python package `ou-tm351-jl-extensions` [documentation].

3.1 Launcher Buttons

Buttons for creating new notebooks and, as well as creating a new terminal, are available from the Launcher.

Hint: If you get an error when trying to create a new notebook saying there is No `root file handle` found, click in the file browser, or click on a directory in the file browser, to set a path, and try again.

If the Launcher button still does not work, right click in the file browser and create a new notebook from the *New Notebook* menu option.

Buttons for launching applications used in your module should have been added to the Launcher, Figure 3.2.

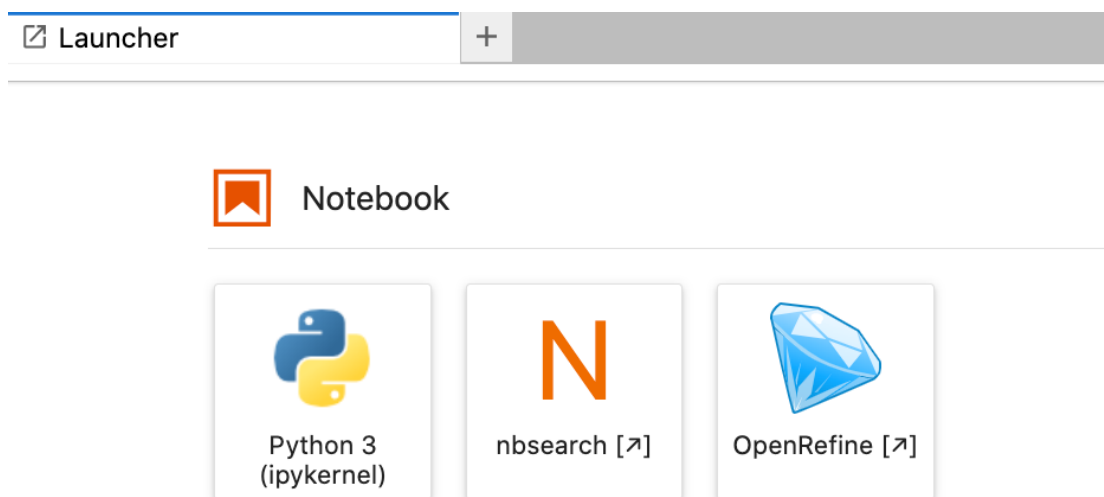


Figure 3.2: The JupyterLab application and new document launcher. *Options may vary by module.* Screenshot showing three launcher buttons: Python3 (ipykernel) notebook, nbsearch and OpenRefine.

USING JUPYTER NOTEBOOKS

Jupyter notebooks are interactive documents that combine editable “content” cells, which may contain structured content written using markdown formatted text, executable “code” cells and executed code cell outputs within a single document.

Notebook code is executed against a notebook kernel. In OU VCEs, notebook kernels are typically provided by self-contained, pre-configured language environments that contain all the required packages necessary for running the module’s coding activities.

4.1 Managing notebooks

Notebooks can be created from the JupyterLab *Launcher* menu, from the main `File > New > Notebook` menu, or by right-clicking in the file browser and selecting `New Notebook` from the pop-up menu.

By default, files are created with a default `UntitledN.ipynb` filename. Notebook files can be renamed by right-clicking on the file in the file browser and selecting `Rename`, or clicking on the filename in the JupyterLab notebook and selecting `Rename Notebook . . .` from the pop-up menu,

4.2 Notebook code kernels

When a notebook is started, a process is started within a programming language kernel environment.

In M348, you will be using notebooks with a language kernel.

As long as the kernel is running (or is hibernated if you hibernate your computer), the values of any variables set within that process will be persisted. That is, the current process *state* will be preserved.

To reset the kernel to an initial empty state, click the notebook toolbar restart button (↺), or select the menu item `Kernel > Restart Kernel`.

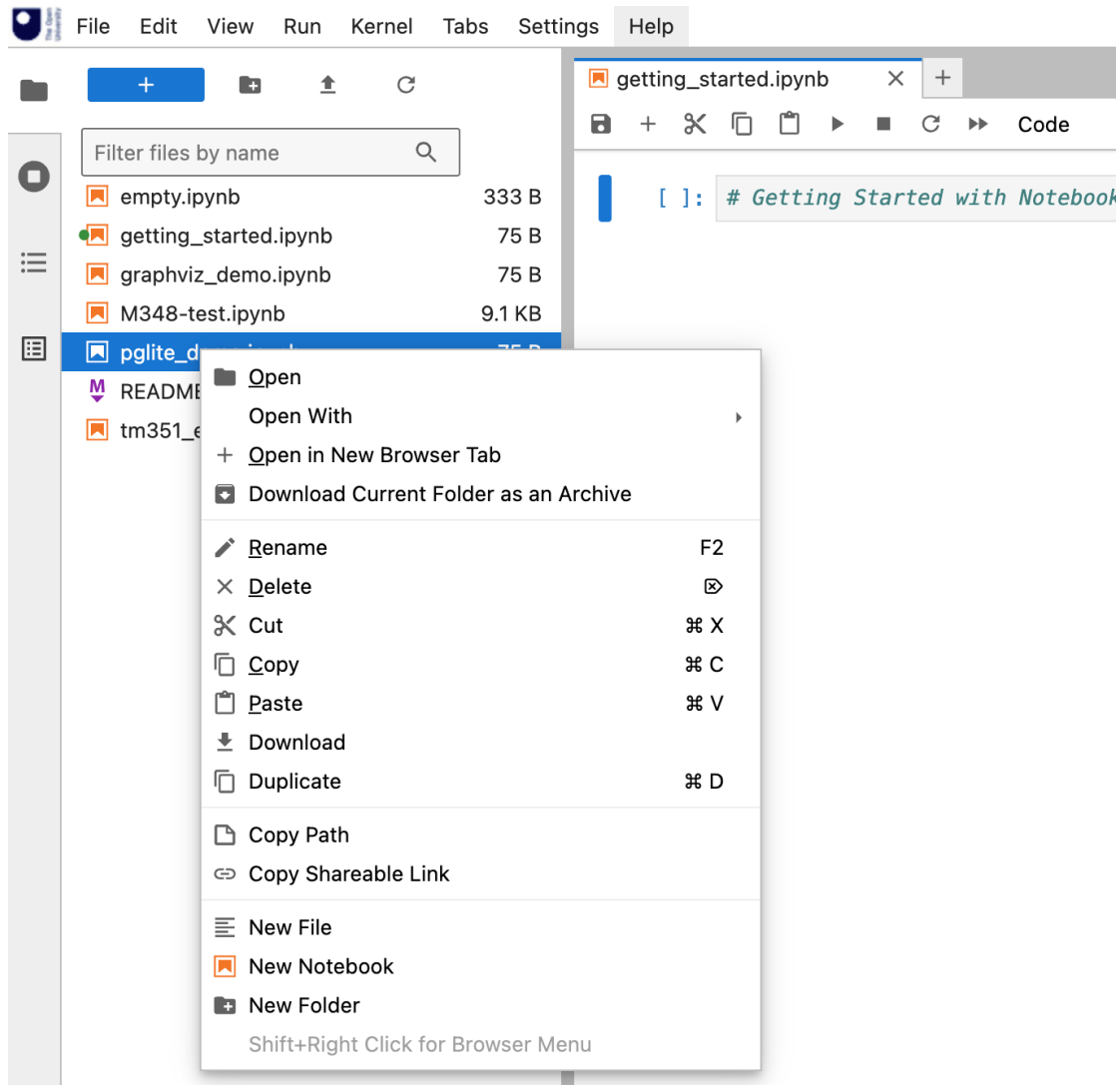


Figure 4.1: JupyterLab file browser, context sensitive menu raised on file
 A screenshot of the context sensitive menu raised from a right-click on a file item in the Jupyter file browser.
 The menu shows actions relating to both the notebook, and the directory the notebook is in.

4.3 Working with code cells

A new Jupyter notebook is created with a single code cell as a default.

Click in the cell to select and edit it, and then run the cell and move on to the next cell either by the keyboard command — `shift-Enter` — or by clicking the play button (▶) in the notebook toolbar.

On running a code cell, outputs from any `print()` or `display()` statements, as well as the value of any object returned from the last executable code line, will be displayed as code cell output, [Figure 4.2](#)

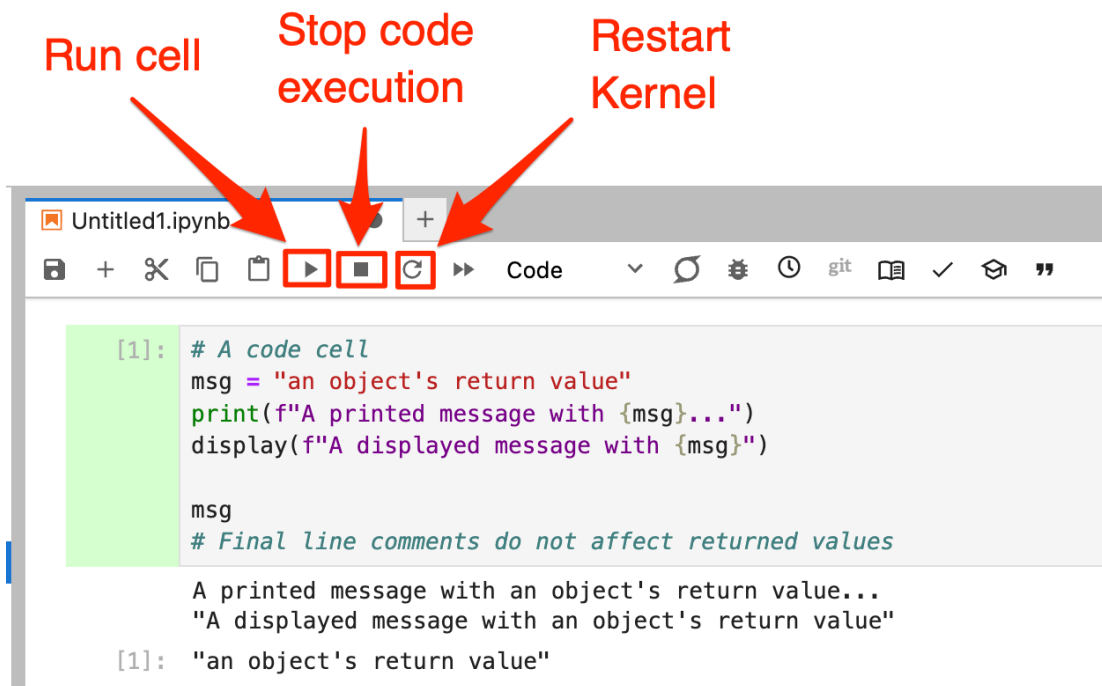


Figure 4.2: A screenshot of an executed notebook code cell with lines of code demonstrating the use of `print` and `display` commands for rendering code cell output, as well as the display of the value of the object returned by the last line of code in the cell. The play button in the notebook toolbar is also highlighted.

You can clear an individual cell's output by right-clicking on the cell and selecting `Clear Cell Output` from the pop-up menu. Clear the output on all cells by selecting `Clear Outputs of All Cells`.

If your program code “hangs” when you try to run it, you can stop the code execution by clicking the stop button (■) in the notebook toolbar or from the `Kernel > Interrupt Kernel` menu option.

4.3.1 Running shell commands from a code cell

In a Python/IPython kernel code cell, you can run a shell command by prefixing the command with a `!`. For example, show the current directory by running `! pwd`.

4.3.2 Code cell execution status indicators

The `jupyterlab_cell_status_extension` provides visual and/or audible indications of the cell run status. The visual indications shown in [Figure 4.3](#) highlight a successfully run cell, a run cell that resulted in an error and a queued/currently running cell. An optional animated “cell flash” effect highlights a code cell that has just completed execution.

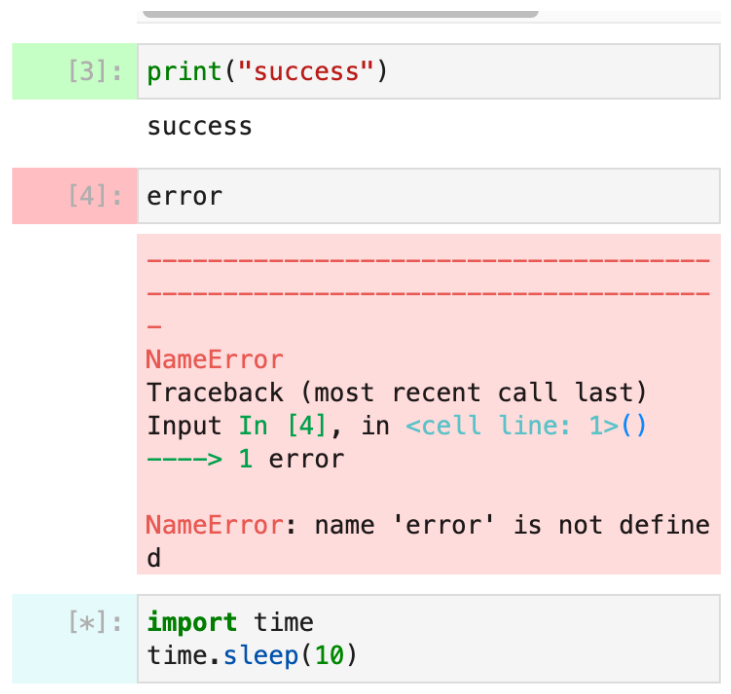


Figure 4.3: Cell status indications

Screenshot showing code cells with different cell run status indications: green (success), red (failure), light blue (awaiting execution).

This extension also supports a range of audible alerts that signal successful or unsuccessful cell execution, as well as spoken error messages.

For further information the audio accessibility features, as well as guidance on changing the settings, see [Section 7 Jupyter Notebook Accessibility](#).

4.4 Working with Markdown cells

When a cell is run, a new code cell is created beneath it by default. You can change a code cell to a markdown cell using the keyboard shortcut `ESC-M` or by changing the cell type from *Code* to *Markdown* in the notebook toolbar.

In edit mode, you can write markup text using the simple Markdown language. Your VCE may also support use of richer *Myst* markdown syntax.

4.4.1 Simple Markdown syntax

Markdown is a simple text based mark-up language for writing richly formatted text, using simple text conventions to identify the formatting you want to apply to the text.

For example:

- *italicised text* is identified by wrapping the text in `*` characters: apply some `*emphasis*`;
- **strong** or **bold** format can be applied by using a double underscores to wrap the text: apply `__strong__` emphasis;
- we can also **combine the two**: apply `__*combined strong and emphasis*__` markup.

List items can be created by prefixing each line item with a `-` at the start of the line. Sublist items can be created indenting a list item.

Headers can be included in a markdown cell by prefixing text with one or more `#` signs, corresponding to the cell heading level (for example, `##` A level 2 heading). If a markdown cell starts with a heading, a collapsible cell indicator will be displayed in the rendered cell view that allows you to collapse all cells underneath that heading cell up to the next markdown cell that starts with the same or higher level heading.

The Jupyter markdown editor will preview the styling that will be applied by the use of markdown tags, where possible.

In markdown cell edit mode, markdown text and code comments are also passed through a spell-checker that highlights words that contain spelling errors.

Disabling the spellchecker

The spellchecker cannot be disabled, but can be configured so that it is unlikely to be called. From the `Settings > Settings Editor` menu, find the spellchecker extension, then set the *Time delay before spellchecking starts (debouncer)* to a big number (the default setting is 200ms).

You can view the rendered form of a markdown cell by “running” it in the same way that you would run a code cell, using `SHIFT-ENTER`, or from clicking the notebook play button.

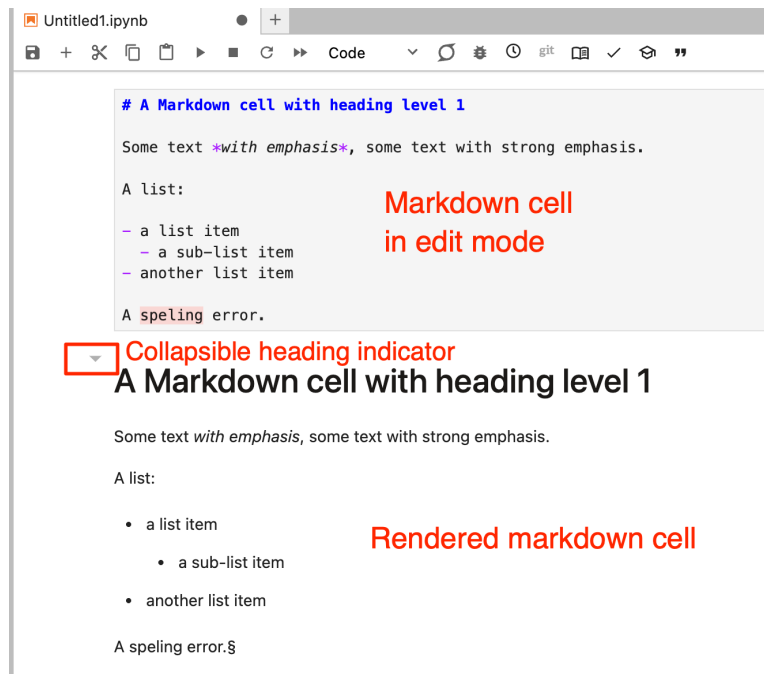


Figure 4.4: Notebook markdown cells

Screenshot showing markdown cells in edit and rendered mode.

The cells include a level 1 heading, emphasised and strong text elements, and list and sublist items.

Double click on a rendered markdown cell to return it to the edit mode.

Inline code and code blocks

You can use inline code style by wrapping the text in single backticks: here is some ``inline code``.

You can also include richly formatted code blocks:

```
# Here is some Python code
def my_function(message):
    """ A hello world function."""
    print(f"The message is: {message}")
```

Wrapping code in triple backtick code fences and identify the language sensitive code styling styling you want to apply (for example, python, R, bash, text):

```
```python
Here is some Python code
def my_function(message):
 """ A hello world function."""
```

(continues on next page)



(continued from previous page)

```
print(f"The message is: {message}")
...
```

## Mathematical notation

You can use LaTeX style syntax to describe and render mathematical equations.

For inline expressions, such as  $E = mc^2$ , wrap the notation in single  $\$$  characters:  $E = mc^2$ .

For block expressions:

$\int_0^\infty e^{-x^2} dx = \frac{\sqrt{\pi}}{2}$

wrap the code using  $\$$  fences:

```
$$
\int_0^\infty e^{-x^2} dx = \frac{\sqrt{\pi}}{2}
$$
```

## Mermaid diagram descriptions

Jupyter notebook markdown cells also support text generated diagrams using [mermaid.js](#) scripts:

```
```mermaid
graph LR;
  A-->B;
  A-->C;
  B-->D;
  C-->D;
  D-->E;
```
```

The rendered cell then displays the corresponding mermaid rendered image, [Figure 4.5](#):

Being able to *write* diagram descriptions within a markdown cell that are then automatically rendered provides an accessible, text-based way for creating (and editing) diagrams. It removes the need for graphical image editors and can simplify the process of diagram creation. Access to the raw “source code” of the diagram also allows tutors to modify or extend diagrams, as well as easily create and share their own diagrams back with students. See the [mermaid.js documentation](#) for a full description of available diagram types.

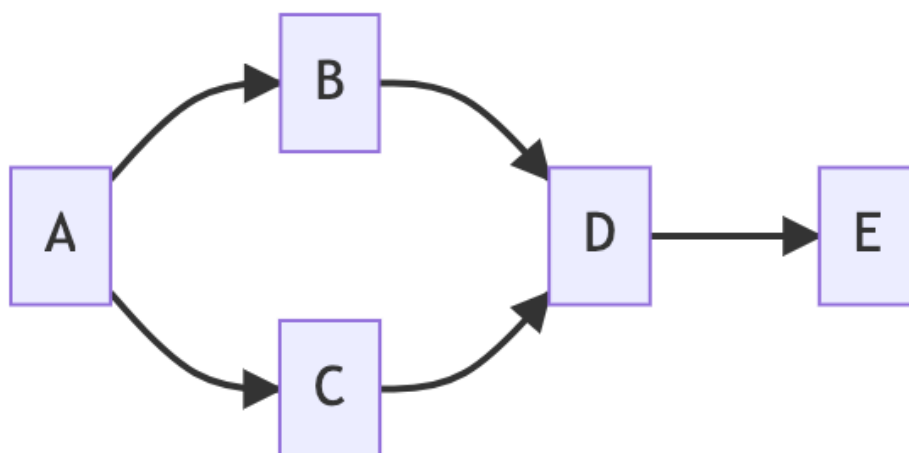


Figure 4.5: Example of a rendered mermaid diagram.

Screenshot showing a simple box and arrows chart flowing left to right. A box labeled A is connected by lines and arrows to two boxes, B and C. Boxes B and C are then connected to box D, which in turn is connected to a box E.

#### 4.4.2 Rich MyST syntax

As well as simple MyST syntax, support is also provided in markdown cells for enriched MyST flavoured markdown syntax [\[docs\]](#).

For example, the following MyST styled admonition block will be rendered as shown in [Figure 4.6](#):

```

```{warning}
This is a warning block.
```

```

Some markdown text...

Warning

This is a **warning** block.

Some *more* markdown text...

Figure 4.6: Example of a warning admonition

Screenshot showing how a triple backticked warning block is rendered with a warning icon and a cream coloured border between two otherwise unstyled markdown blocks.

Other blocks include `danger` (red header bar), `note` (blue), `seealso` (lilac), `important` (light grey-green) and `tip` (light green). The header bars also carry distinguishing leading icons. Using the `{admonition}` My Title style block, a title can be added to the block and styled using

the appropriate admonition type set as a `:class:` value. For example, the following block will be rendered as shown in [Figure 4.7](#)

```
```{admonition} Take this as a warning!
:class: warning

This is a warning block.
```
```

Some markdown text...

▲ Take this as a warning!

This is a warning block.

Some *more* markdown text...

Figure 4.7: MyST syntax admonition block with a title, styled as a warning, between two otherwise unstyled markdown blocks.

## 4.5 Coloured cell backgrounds

Many of the notebooks used in OU modules used coloured cell backgrounds to indicate different types of content within a notebook, using four thematically coloured background cells, as shown in [Figure 4.8](#). Cell backgrounds are persistent and are toggled from notebook toolbar buttons.

- *activity* (blue): cells that describe activities or exercises;
- *learner* (yellow): cells that students are expected to modify as part of their learning or assessment.;
- *tutor* (pink): important information, or text added as feedback by a tutor on assessed material;
- *solution/success* (green): used to indicate a worked solution or successful outcome.

The toggle buttons can be individually enabled / disabled; the colours applied to each cell type are also user customisable via user settings as shown in [Figure 4.9](#).

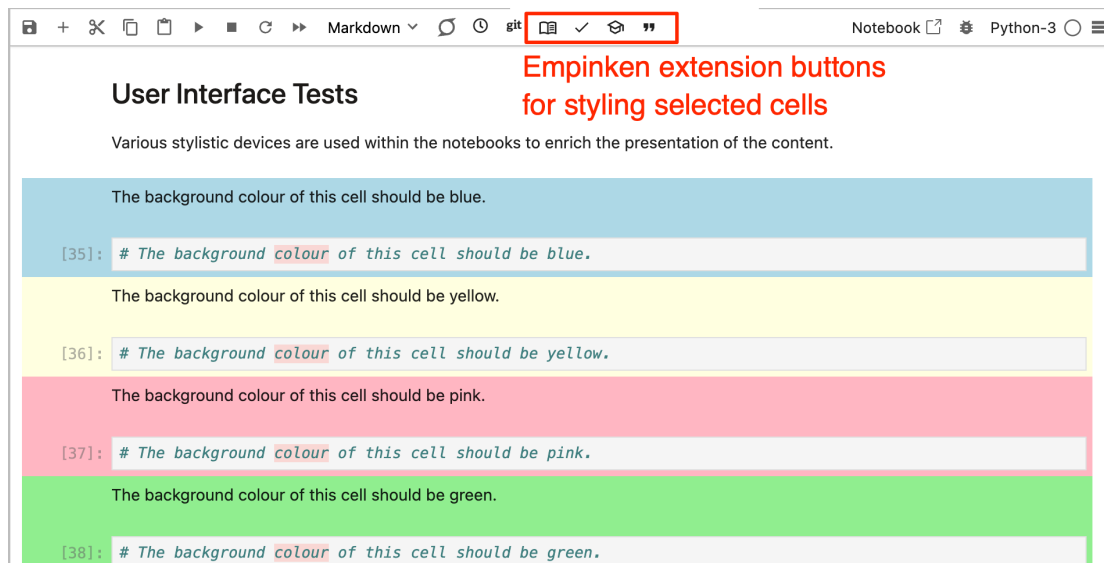


Figure 4.8: A screenshot of “empinken” extension coloured background cells, used to identify different cell roles.

Four coloured cell types (both markdown and code cells) are shown: blue, yellow, pink and green.

## 4.6 Navigating notebooks

The Jupyter environments provide a dynamically generated table of contents listing for a selected notebook from the left hand sidebar palette, [Figure 4.10](#).

*In the Jupyter Notebook v7 environment, open the table of contents from the Notebook View -> Table of Contents menu (shift-command-k keyboard shortcut).*

The table of contents offers two main benefits:

- it provides an overview of the whole document and signposts key, headed elements within it;
- it provides an effective way of navigating to different parts of the document.

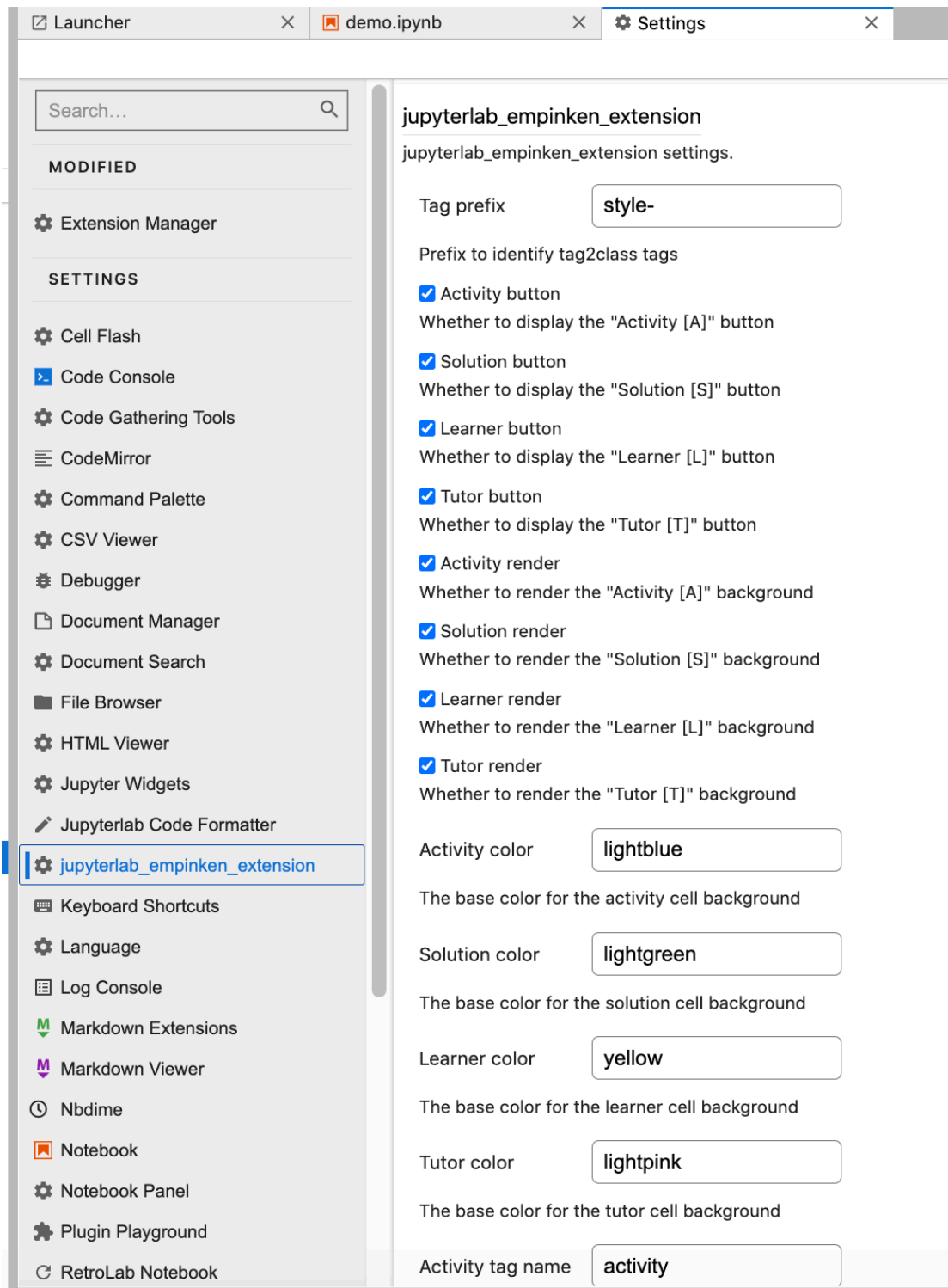


Figure 4.9: Screenshot of empinken extension user settings

Options are shown that allow a user to control whether a toolbar button is displayed, whether particular cell backgrounds are rendered, and the colour used to render each background.

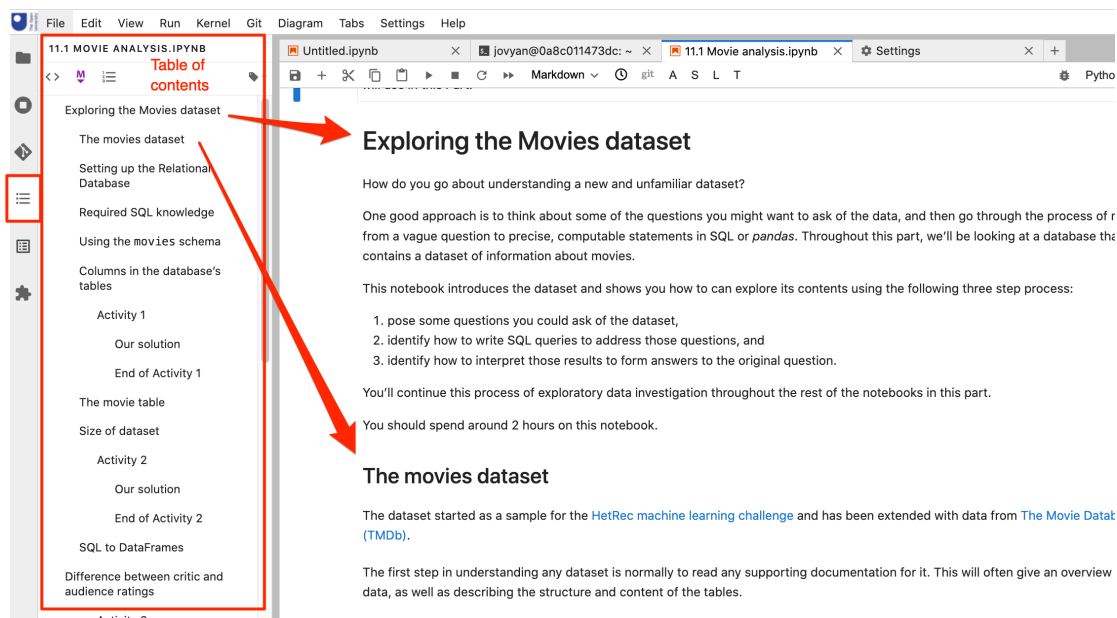


Figure 4.10: JupyterLab notebook table of contents  
Sidebar showing a table of contents navigation tool generated from the headings in a selected open notebook.

# **Part III**

## **Troubleshooting**





## **ADDITIONAL SUPPORT**

If you encounter any technical problems working the VCE or accessing the services that it provides, or have problems running provided code or your own code, the first thing to remember is **don't panic**: in many cases, you may be able to fix the problem yourself; in others, a readily available helping hand should be able to resolve your problem.

*If your problem is more related to your personal learning, understanding or the assessment of the module, it would be appropriate to contact your tutor for advice.*

In general, you may find it useful to address technical issues by working through the problem in the following way:

1. Check any error messages, if appropriate. Error messages often contain a clue as to what caused them. If you are using a terminal to issue commands, or writing computer code, a common problem is a command not being recognised because of a typographical error. Computers are very literal in what they do, so a misplaced comma or a single character out of place may be all that needs fixing.
2. Check the module forums to see if there are any official announcements relating to the same problem, or whether any other students have posted similar issues. Many forums are moderated and the forum moderators will answer any questions they can, or pass them on to the module team if they can't. Appropriate responses will then be posted to the forum.
3. If a technical problem similar to yours has already been reported, but the suggested solution does not work for you, reply to the suggestion. In your reply, explain what you tried and what did or did not happen, including any error messages, if available, either as copied text or as a screenshot. Provide details of your operating system, if relevant. Sometimes you might find that trying to explain the issue identifies the solution. In such a case, if you think the problem may be a common one, you may want to post a message to the forum explaining the error you encountered and how you fixed it.
4. If your problem has not already been identified, post a message to the 'Jupyter notebooks' forum. Give the message a clear title that identifies the problem. Include in the body of the message a description of what you did and what did or did not happen, including any error messages, if available, either as copied text or as a screenshot. Once again, provide details of your operating system and/or browser, if appropriate. As before, you may find that trying to

explain the problem reveals the solution. If you think that sharing the issue — and the solution you discovered — may be of some help to others, please do so via the appropriate forum.

5. If you are struggling with identifying what the problem is and would find it useful to talk to someone, get in contact with:

- the [OU Computing Helpdesk](#), for general IT and software installation enquiries
- your tutor, by phone or email or other agreed contact method for issues specifically about taught content and practical activities within the module.

To give yourself the best chance of resolving any issue in a timely fashion, *plan ahead*. The assignments require you to make use of the VCE, so don't leave it until the last minute to try the VCE for the first time.

More specific guidance for working through issues related to various software problems can be found in [Troubleshooting](#).

## **TROUBLESHOOTING**

While we have tried to ensure that the software installation process and all the software-related activities run smoothly, there is always a chance that you may encounter a problem or issue with the software.

The section *Additional support* describes a general strategy for working through problems or raising technical issues. This section describes more specific guidance for working through issues with particular software elements.

---

### **Optional content**

You shouldn't need to work through the following unless there are specific problems that you have encountered when working with the VCE.

---

## **6.1 Accessing a terminal command-line interface within the VCE**

For many modules, you should not need to access the VCE command line.

However, if you find you do want to issue a command-line command or gain access to a command-line interface or terminal within the virtual machine, there are three main ways of doing this from inside the container:

- Within a notebook code cell, in the first line of the cell enter the command:

```
%%bash
```

Any additional code lines you enter into the cell will be interpreted as shell commands and executed as such when you run the cell.

- Within a notebook code cell, prefix the command-line command you want to run with an exclamation mark (!). For example, to list the contents of the current directory, run the Linux/Unix `ls` command:

```
! ls
```

- Use the Jupyter interactive shell. From the notebooks folder home page, create a new terminal by selecting *Terminal* from the *New* drop-down menu on the right-hand side as shown in [Figure 6.1](#).

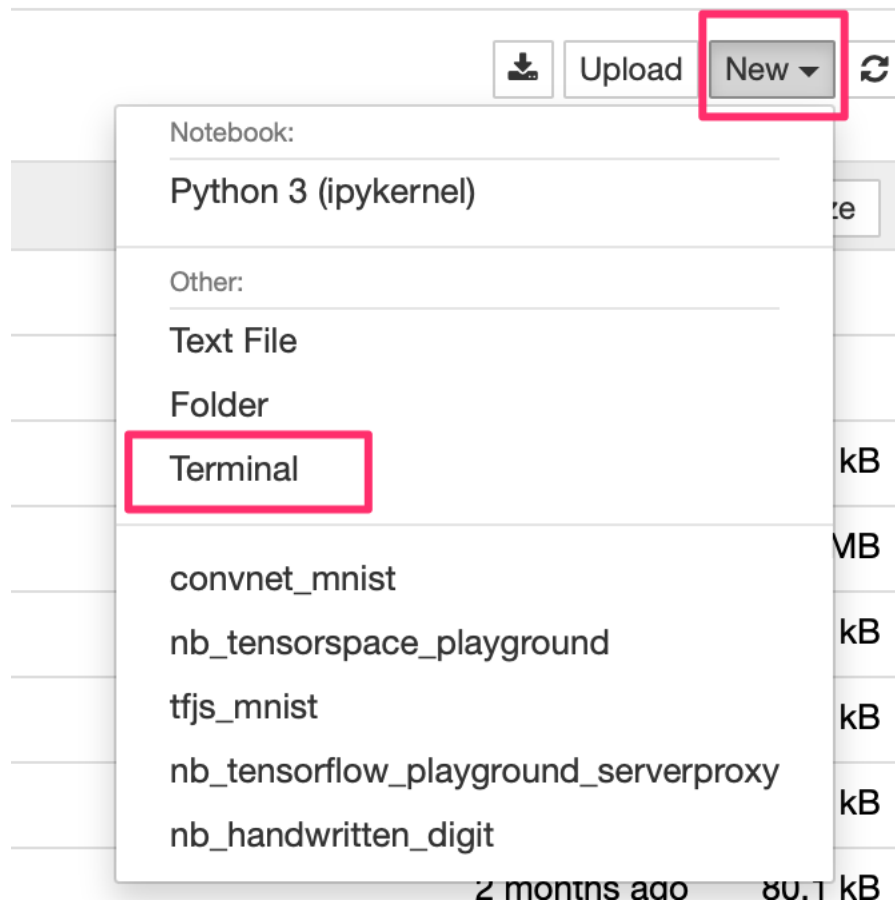


Figure 6.1: The “New” drop-down menu

A screenshot of part of the notebooks folder home page showing three buttons — ‘Download as zip’ (indicated as a down arrow above a horizontal bar), ‘Upload’ and ‘New’ — and a refresh button. The ‘New’ button has been pressed, revealing a drop-down menu with various items’. The item ‘Terminal’ has been highlighted.

See the section for more information on commands that you can run from the command line inside the VCE container.

## 6.2 Problems arising from working with large notebooks

If you have tried to display a large amount of data in a notebook cell then you may find that your browser struggles to display the notebook.

You should first attempt to clear all the output cells using the notebook menu *Cell > All Output > Clear*. If you are entirely unable to run the notebook to access the menu, first try closing any other notebooks you have open then try opening the problem notebook. If that doesn't allow the notebook to open, then using a command-line command you can run the problem notebook through a separate process that will clear all the cell outputs.

Open a terminal in the VCE. Then change directory (`cd`) to the shared folder, and issue the following command (all on one line):

```
jupyter nbconvert --to notebook --ClearOutputPreprocessor.enabled=True ↵
YOURNOTEBOOK.ipynb
```

By default, the clean notebook will be named `YOURNOTEBOOK.nbconvert.ipynb`. To clean the notebook and retain the same filename, add the flag `--inplace` to the command line:

```
jupyter nbconvert --inplace --to notebook --ClearOutputPreprocessor.
↵enabled=True YOURNOTEBOOK.ipynb
```

The notebook server may also run into memory problems if you have a large number of notebooks open. Try stopping all the notebooks and then restarting the notebook you are currently working on. (You will need to rerun the code cells to restore the state of the variables.)

## 6.3 Problems associated with running out of memory

If in a long running container, if you close a notebook but do not stop the associated kernel, you may end up with a large number of still running kernels even if you do not have any notebook tabs open in your browser. You can find a list of running notebooks from the *Running* tab on the notebook homepage.

You can also see an indication of which notebooks still have running kernels associated with them when viewing a file listing on the notebook home page, [Figure 6.2](#).

If you do find you have a lot of running kernels without associated open notebooks, stop them from the *Running* tab.

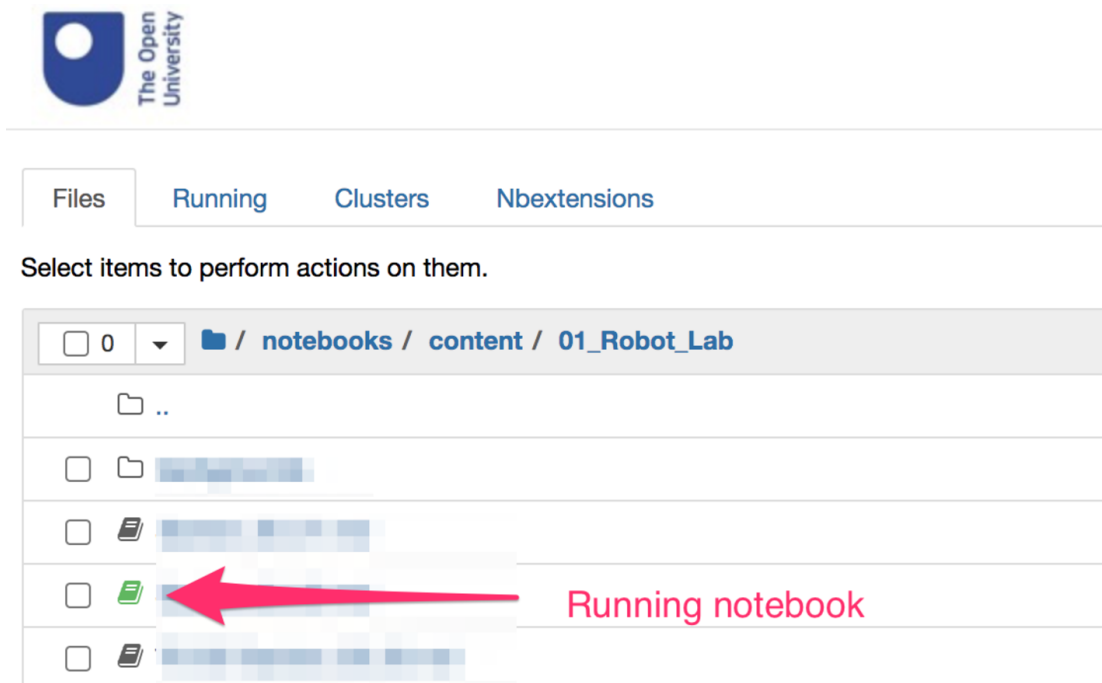


Figure 6.2: An example of the notebooks folder home page. The actual directory file path and file listing is indicative only.

A screenshot of a web browser showing the 'Files' tab of the Jupyter notebook environment. A number of items are shown pixellated, but the one with a green icon has an arrow pointing to it labelled 'Running notebook'.

## 6.4 Problems running notebook code

If you have problems running code in the module notebooks, or running your own code, read any error messages carefully to see if they point to the cause of the problem. You may also find it useful to check the relevant documentation, or run a web search using key elements of any error messages that occur.

Technical Q&A sites such as [Stack Overflow](#) contain answers to a wide variety of ‘*How do I ...?*’ style questions and are often well worth exploring. Results from searches on Stack Overflow can also be limited to relevant questions by adding appropriate programming language tags to your query, such as *python* or *R*.

Generative AI tools may also help answer your question. They are particularly good at providing small amounts of sample code to get you started. Microsoft’s [Bing copilot](#) provides access to a free model and is easy to use. Google’s [Gemini](#), OpenAI’s [ChatGPT](#) and Anthropic’s [Claude.ai](#) also offer access to free models but require registration to use them. If you use a generative AI tool as part of your work on an assignment you must cite and reference it appropriately, and include a copy of the output in an appendix.

The OU provides guidance on how to use generative AI as part of your studies here: <https://about.open.ac.uk/policies-and-reports/policies-and-statements/gen-ai/generative-ai-students>

Do not rely solely on the AI output for your assignments, as they frequently produce incorrect answers, partial answers, and answers that are not well tailored to the case study in the question

If you continue to have problems, check the module forums to see if anyone else has encountered — and resolved — the same issue. You may also ask for support in module forums or from your tutor.

The OU Computing Helpdesk will not be able to help with detailed technical queries arising from the module practical activities, so please limit any module-content related enquiries to the module forums or your tutor. The OU Computing Helpdesk can help with more general computing matters, including some support for installation problems that might occur.

## 6.5 Finding version numbers for software in your VCE

There may be occasions, such as when troubleshooting or debugging problems, when you are asked to confirm which versions of software applications you are running or programming packages you have installed in your VCE. You will be provided with guidance on how to find the required version numbers.

If you are running into repeated problems with an environment inside a local VCE, report the digest number for the image used to generate the container. You can find this by running the command `docker images --digests` from the command line on your computer. Look for the line corresponding to the `ousefulcoursecontainers/ou-m348:24j` image: the digest is the number starting with the prefix: `sha256:.`





# **Part IV**

## **Productivity**



## JUPYTER NOTEBOOK ACCESSIBILITY

The Jupyter environments, and the Jupyter notebooks contained within them, are rendered within a browser as HTML. Instructional text in notebook Markdown cells should be directly readable by screen readers. Code cells are contained within HTML group elements and may need to be intentionally navigated to.

Most of the Jupyter notebook features are keyboard accessible. Several pre-installed extensions provide further support in terms of visual styling and limited audio feedback support.

If you struggle to use the VCE for any reason, including but not limited to incompatibility with any tools you may use to improve software access or usability, please raise an issue in the module forums or contact your tutor.

### 7.1 Keyboard shortcuts

The Jupyter notebook interface supports a wide range of pre-defined keyboard shortcuts to menu and toolbar options ([official docs - command list](#)). The shortcuts can be displayed using the Keyboard Shortcuts item from the notebook *Help* menu or via the `Ctrl-shift-H` (Windows) / `Shift-command-H` (Mac OSX) keyboard shortcut [Figure 7.1](#).

You can view and edit existing keyboard shortcuts via the *Settings -> Settings Editor -> Keyboard Shortcuts* display.

To change or add a particular keyboard shortcut for a specific command, click the entry in the “Shortcut” column for that command and enter the key-presses that should be used as the keyboard shortcut for it.

## Keyboard Shortcuts

|                           |                  |
|---------------------------|------------------|
| Close Tab                 | Alt + W          |
| Close and Shut Down       | Ctrl + Shift + Q |
| Activate Next Tab         | Ctrl + Shift + ] |
| Activate Next Tab Bar     | Ctrl + Shift + . |
| Activate Previous Tab     | Ctrl + Shift + [ |
| Activate Previous Tab Bar | Ctrl + Shift + , |
| Show Left Sidebar         | Cmd + B          |
| Simple Interface          | Shift + Cmd + D  |
| Show Right Sidebar        | Cmd + J          |
| Activate Command Palette  | Shift + Cmd + C  |
| Show Keyboard Shortcuts   | Shift + Cmd + H  |

Close

Figure 7.1: The JupyterLab ‘Keyboard shortcuts’ dialogue box  
The JupyterLab ‘Keyboard shortcuts’ dialogue box showing some of the available keyboard shortcuts.

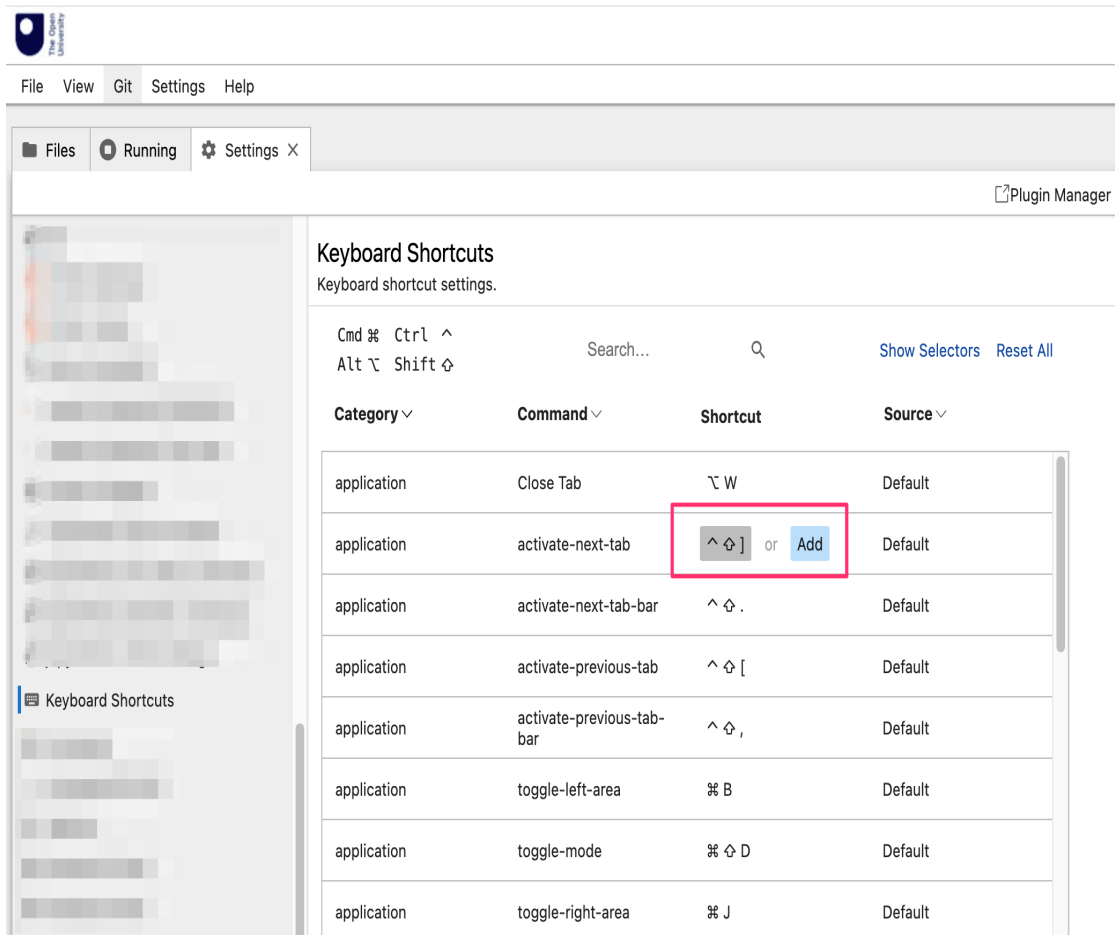


Figure 7.2: The JupyterLab *Settings* -> *Settings Editor* -> *Keyboard Shortcuts* editor

The JupyterLab *Settings* -> *Settings Editor* -> *Keyboard Shortcuts* editor showing some of the available commands. Clicking in the *Shortcut* column for a particular command allows you to edit the current keyboard command, and/or add additional keyboard shortcuts.

## 7.2 Visual Display Settings

A wide range of visual display settings can be set via the *Settings* -> *Theme* menu.

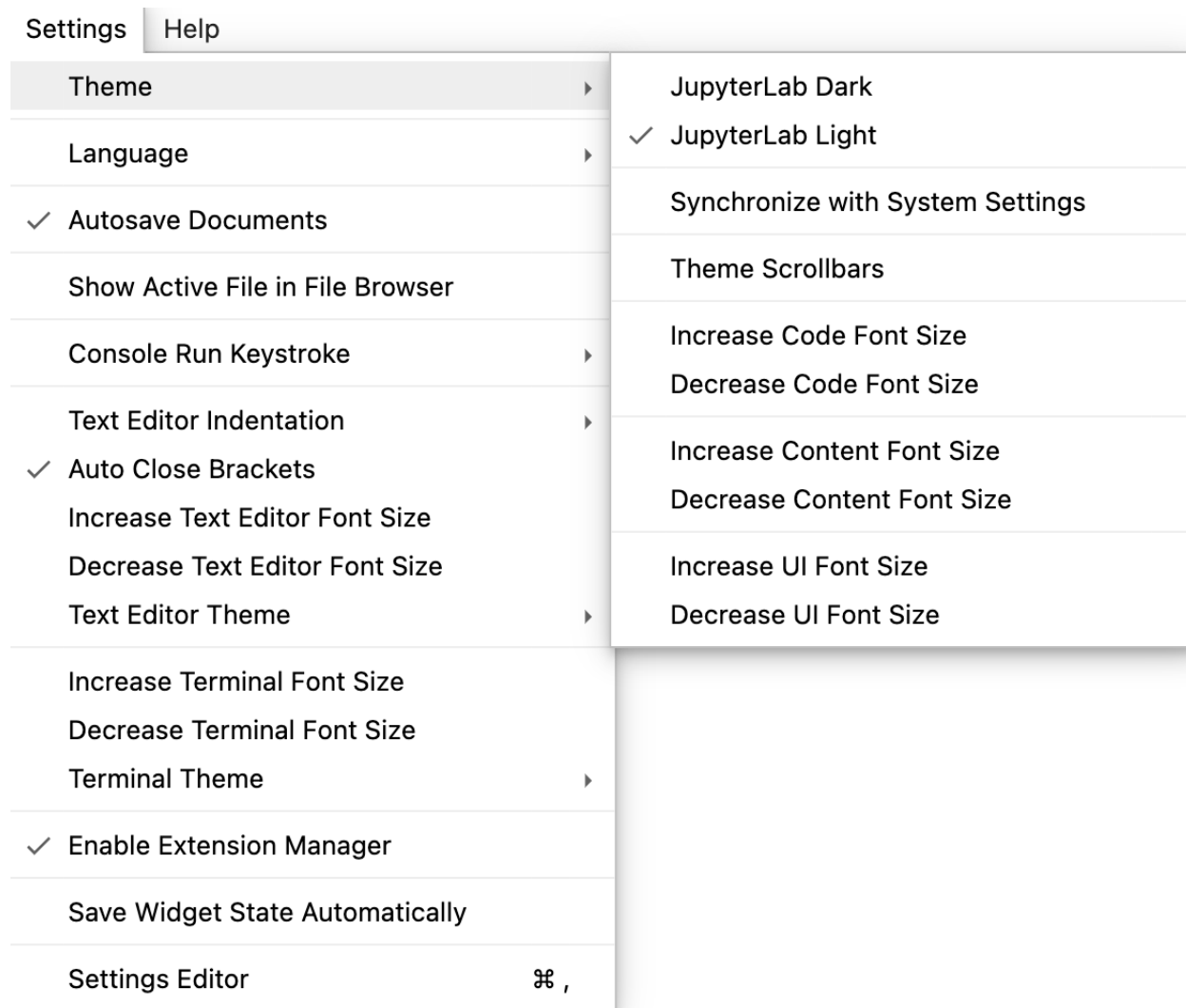


Figure 7.3: The JupyterLab “Settings -> Theme” menu

The JupyterLab “Settings -> Theme” menu, showing a range of available settings, including theme selection, code, content and UI text font size display, and language pack.

By default, the Jupyter environment is displayed using a simple black on white theme (*JupyterLab Light*). A “dark” theme (*JupyterLab Dark*) is also available.

The menu also provides a means of increasing or decreasing the font size for user interface elements, notebook content (markdown cells) and notebook code cells. Changes to the font size are saved in a persistent settings file (`./jupyter/lab/user-settings/@jupyterlab/apputils-extension/themes.jupyterlab-settings`).

The *Settings* -> *Language* menu provides access to alternative user interface language packs. By default, language packs for English (default), French and Chinese are preinstalled.

**Hint:** If you would like to request additional language packs to be installed by default into your module VCE, please contact your module team via the module forums.

## 7.3 Audible Alerts

Various tools have been pre-installed into the VCE that can optionally provide audible alerts and spoken feedback associated with code cell execution.

### 7.3.1 Cell execution status

The preinstalled `cell execution status extension` provides an enhanced visual display of the run state of a code cell:

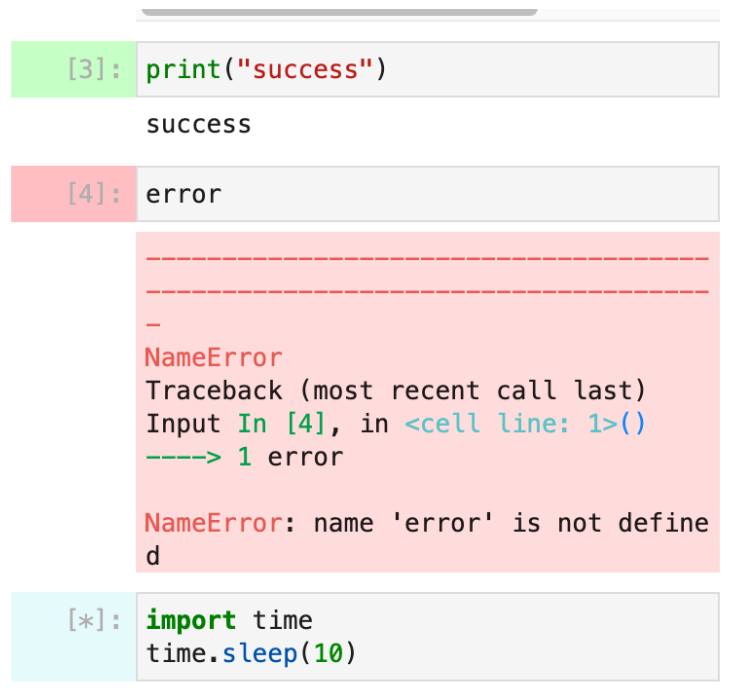


Figure 7.4: Cell status indications

Screenshot showing code cells with different cell run status indications: green (success), red (failure), light blue (awaiting execution).

A cell flash effect can be optionally enabled to highlight when a cell has finished executing.

Audible alerts can also be enabled from the *Settings* -> *Settings Editor* panel that identify when cells have successfully or unsuccessfully completed their execution. Further settings enable spoken outputs for cell execution error messages.

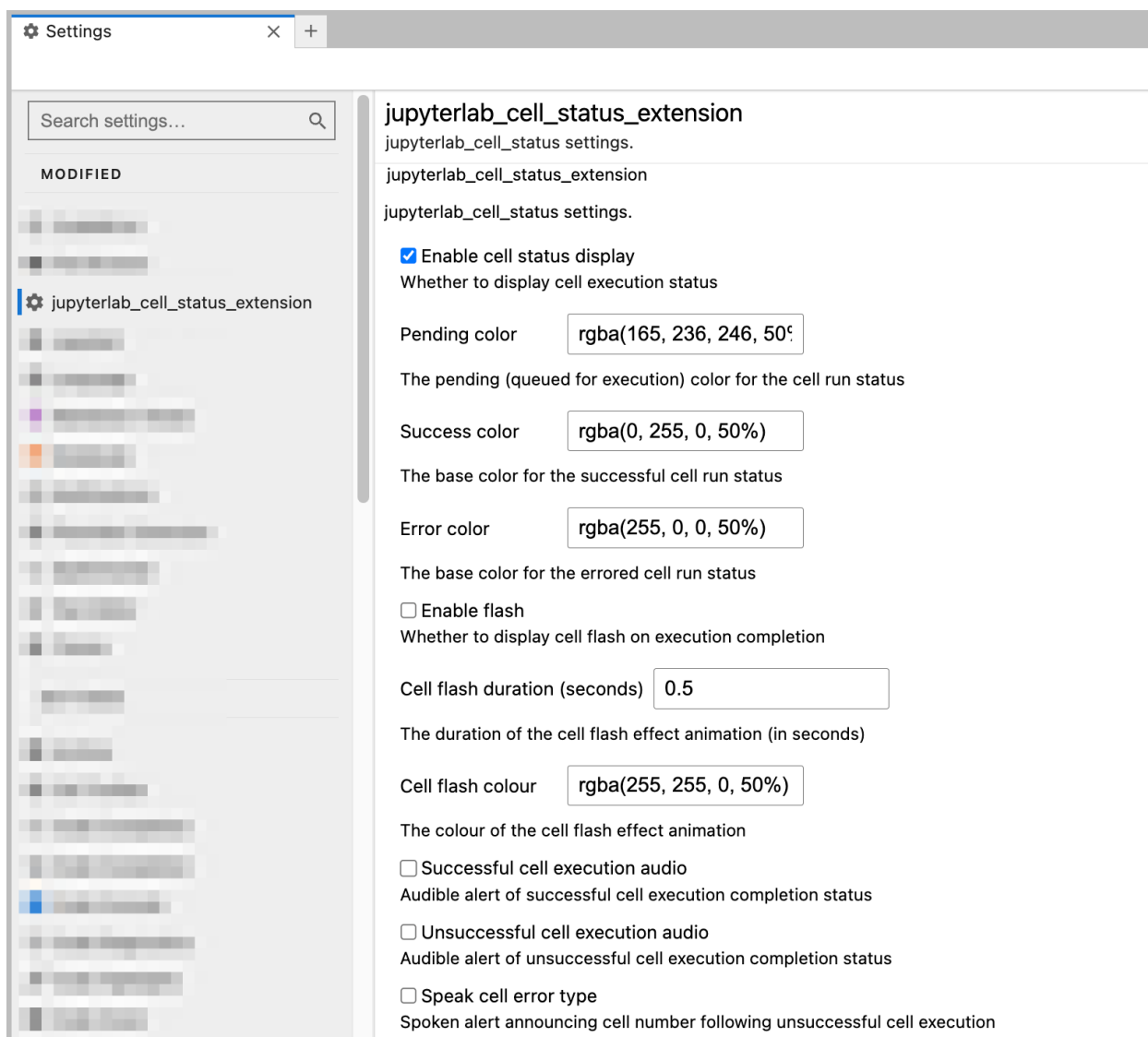


Figure 7.5: The JupyterLab “cell execution status” extension settings

The JupyterLab “cell execution status” extension settings panel, showing a range of available settings, including the ability to enable/disable cell status indication, cell flash on execution, audible cell execution completion status alerts and spoken cell execution error messages.



## 7.3.2 Audible Logging

One of the simplest ways of debugging code execution in *ad hoc* way is display a log or “print debug” message at certain points in your code.

The preinstalled `ou-logger-py` Python package builds on the Python `logger` package to allow you to display messages at various priority levels, or have those messages spoken aloud using your browser’s built in speech synthesis package.

In the first code cell of a notebook, import the logger as `from ou_logger import logger, set_handler`

This will display an information message:

```
Logger enabled. Set level as: logger.setLevel(LEVEL), where LEVEL is
one of: DEBUG, INFO, WARNING, ERROR (default), CRITICAL.
Set text and/or text-to-speech output: set_handler('text, tts')
Usage: e.g. logger.error('This is an error message')
```

Logged messages can printed, spoken aloud using the browser text-to-speech (TTS) engine, or both.

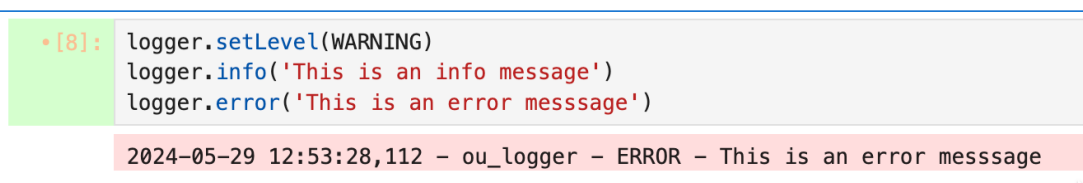
Enable text and/or TTS output by adding the relevant line under your import statement at the top of the notebook.

```
set_handler("text") # Just text output
set_handler("text, tts") # Text and speech output
set_handler("tts") # Just speech output
```

Logging messages will be displayed at or above the declared logging level. For example:

- `logger(CRITICAL)` will only display CRITICAL messages;
- `logger(WARNING)` will display WARNING, ERROR and CRITICAL messages.

By default, messages will be displayed as text:



The screenshot shows a Jupyter notebook cell with the following code:

```
•[8]: logger.setLevel(WARNING)
 logger.info('This is an info message')
 logger.error('This is an error message')
```

The output of the cell is displayed on a pink background:

```
2024-05-29 12:53:28,112 - ou_logger - ERROR - This is an error message
```

Figure 7.6: Example ou-logger message.

Example of ou-logger message displayed on a pink background as notebook streamed output



## **FILE MANAGEMENT**

If you are working within a hosted VCE, your files will be saved to hosted online storage and will remain accessible for at least the duration and immediate aftermath of the module presentation.

If you are working in a local VCE environment, you can use a shared folder to mount files directly from your desktop/host environment into the VCE and then save any changes back to the desktop environment.

### **8.1 Uploading and Downloading Files**

To upload files or zipped file archives, click the up arrow (“Upload Files”) icon in the JupyterLab file browser toolbar; *(in the notebook v7 interface, use the `Upload` button from the notebook UI tool bar)*.

To download a compressed file archive of a folder and its contents, in the file browser, right click on the folder and from the pop-up menu select `Download Current Folder as an Archive`.

This works for local and hosted environments and provides a convenient way to save the contents of a working TMA related directory, for example, for use as a submission to the ETMA system.

### **8.2 Zipping and unzipping compressed archive files**

A common way of transporting large files or bundles of files, such as collections of Jupyter notebooks, is to compress them into a single compressed archive file, such as a `.zip` file.

To uncompress a zipped file, from the file browser, right click on the zipped file (e.g. `my_archive.zip`) and select `Extract Archive`.

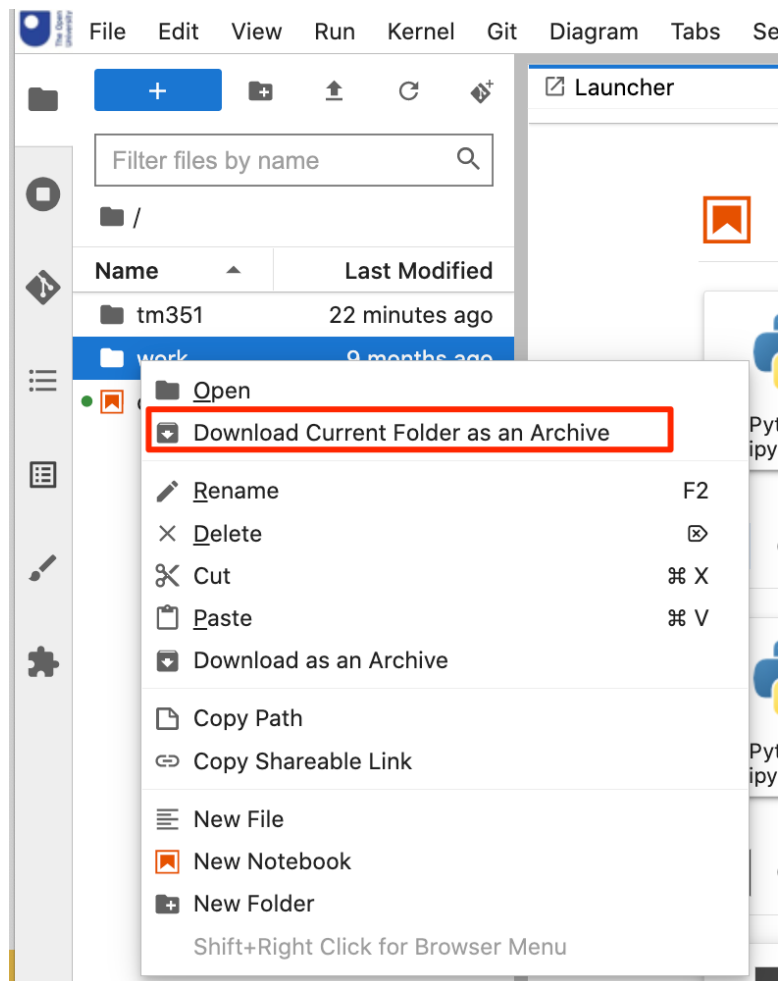


Figure 8.1: Download a folder as a zipped file archive.  
Screenshot of context sensitive menu for file directory, with “Download Current Folder as Archive” option selected.

## 8.2.1 Working with archive files on the command line (advanced)

Archive files can also be created or opened from the command line. Open a terminal using the JupyterLab *Terminal* Launcher button (or in the notebook UI, via the *New* button on the Jupyter Notebook homepage), and then, in the terminal:

- to unzip a file called `filename.zip`, use the command `unzip filename.zip` and press enter: the file will be unzipped into the current directory. *Depending on the contents of the zip file, the unzipped files may be contained in a newly created directory within the current directory.* Close the terminal. (If the `.zip` file is not in the current directory, use the `cd PATH` command to change directory to the required location before running the `unzip` command, or specify the path to the file (`unzip PATH/TO/filename.zip`).
- to zip a directory `./my_directory` in the current directory, and all that directory's contents, use the command: `zip -r my_directory_archive.zip my_directory/`.

## 8.3 Advanced file management

As well as manually uploading and downloading files, or mounting shared local folders in the local VCE, the following extension(s) are available installed in the M348 VCE. Please refer to the official extension documentation or the module forums for further information.

### 8.3.1 Local file system access

It is possible to open files directly from your personal machine within the Jupyter environment (hosted or local) by mounting files from the desktop file system into the browser.

See `g-local_fs.md#local-file-system-access` for more details.

### 8.3.2 Syncing files to a GitHub repository

You can synchronise files within the VCE to and from a personal GitHub repository. *Note that you should ensure the repository is a **private** repository so that you do not unwittingly share any of your assessment work in public.*

See `g-jupyterlab_git.md#using-git-and-github-in-jupyterlab` for more details.