Jupyter Notebook Documentation

Release 5.0.0.dev

https://jupyter.org

April 07, 2016

What's New in Jupyter Notebook

Release *4.1.0*

Release Announcement
- Cell toolbar selector moved to View menu
- Restart & Run All Cells added to Kernel menu
- Multiple-cell selection and actions including cut, copy, paste and execute
- Command palette added for executing Jupyter actions
- Find and replace added to Edit menu

To upgrade to the release: `pip install notebook --upgrade` or `conda upgrade notebook`

The Jupyter Notebook

1.1 Introduction

The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results. The Jupyter notebook combines two components:

A web application: a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output.

Notebook documents: a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects.

See also:

See the installation guide on how to install the notebook and its dependencies.

1.1.1 Main features of the web application

- In-browser editing for code, with automatic syntax highlighting, indentation, and tab completion/introspection.

- The ability to execute code from the browser, with the results of computations attached to the code which generated them.

- Displaying the result of computation using rich media representations, such as HTML, LaTeX, PNG, SVG, etc. For example, publication-quality figures rendered by the matplotlib library, can be included inline.

- In-browser editing for rich text using the Markdown markup language, which can provide commentary for the code, is not limited to plain text.

- The ability to easily include mathematical notation within markdown cells using LaTeX, and rendered natively by MathJax.

1.1.2 Notebook documents

Notebook documents contains the inputs and outputs of a interactive session as well as additional text that accompanies the code but is not meant for execution. In this way, notebook files can serve as a complete computational record of a session, interleaving executable code with explanatory text, mathematics, and rich representations of resulting objects. These documents are internally JSON files and are saved with the `.ipynb` extension. Since JSON is a plain text format, they can be version-controlled and shared with colleagues.

Notebooks may be exported to a range of static formats, including HTML (for example, for blog posts), reStructured-Text, LaTeX, PDF, and slide shows, via the nbconvert command.

Furthermore, any `.ipynb` notebook document available from a public URL can be shared via the Jupyter Notebook Viewer (nbviewer). This service loads the notebook document from the URL and renders it as a static web page. The results may thus be shared with a colleague, or as a public blog post, without other users needing to install the Jupyter notebook themselves. In effect, nbviewer is simply nbconvert as a web service, so you can do your own static conversions with nbconvert, without relying on nbviewer.

See also:

Details on the notebook JSON file format

1.2 Starting the notebook server

You can start running a notebook server from the command line using the following command:

```
jupyter notebook
```

This will print some information about the notebook server in your console, and open a web browser to the URL of the web application (by default, `http://127.0.0.1:8888`).

The landing page of the Jupyter notebook web application, the **dashboard**, shows the notebooks currently available in the notebook directory (by default, the directory from which the notebook server was started).

You can create new notebooks from the dashboard with the `New Notebook` button, or open existing ones by clicking on their name. You can also drag and drop `.ipynb` notebooks and standard `.py` Python source code files into the notebook list area.

When starting a notebook server from the command line, you can also open a particular notebook directly, bypassing the dashboard, with `jupyter notebook my_notebook.ipynb`. The `.ipynb` extension is assumed if no extension is given.

When you are inside an open notebook, the *File | Open...* menu option will open the dashboard in a new browser tab, to allow you to open another notebook from the notebook directory or to create a new notebook.

Note: You can start more than one notebook server at the same time, if you want to work on notebooks in different directories. By default the first notebook server starts on port 8888, and later notebook servers search for ports near that one. You can also manually specify the port with the `--port` option.

1.2.1 Creating a new notebook document

A new notebook may be created at any time, either from the dashboard, or using the *File → New* menu option from within an active notebook. The new notebook is created within the same directory and will open in a new browser tab. It will also be reflected as a new entry in the notebook list on the dashboard.

1.2.2 Opening notebooks

An open notebook has **exactly one** interactive session connected to an IPython kernel, which will execute code sent by the user and communicate back results. This kernel remains active if the web browser window is closed, and reopening the same notebook from the dashboard will reconnect the web application to the same kernel. In the dashboard, notebooks with an active kernel have a `Shutdown` button next to them, whereas notebooks without an active kernel have a `Delete` button in its place.

Other clients may connect to the same underlying IPython kernel. The notebook server always prints to the terminal the full details of how to connect to each kernel, with messages such as the following:

```
[NotebookApp] Kernel started: 87f7d2c0-13e3-43df-8bb8-1bd37aaf3373
```

This long string is the kernel's ID which is sufficient for getting the information necessary to connect to the kernel. You can also request this connection data by running the `%connect_info` magic. This will print the same ID information as well as the content of the JSON data structure it contains.

You can then, for example, manually start a Qt console connected to the *same* kernel from the command line, by passing a portion of the ID:

```
$ ipython qtconsole --existing 87f7d2c0
```

Without an ID, `--existing` will connect to the most recently started kernel. This can also be done by running the `%qtconsole` magic in the notebook.

See also:

Decoupled two-process model

1.3 Notebook user interface

When you create a new notebook document, you will be presented with the **notebook name**, a **menu bar**, a **toolbar** and an empty **code cell**.

notebook name: The name of the notebook document is displayed at the top of the page, next to the `IP[y]: Notebook` logo. This name reflects the name of the `.ipynb` notebook document file. Clicking on the notebook name brings up a dialog which allows you to rename it. Thus, renaming a notebook from "Untitled0" to "My first notebook" in the browser, renames the `Untitled0.ipynb` file to `My first notebook.ipynb`.

menu bar: The menu bar presents different options that may be used to manipulate the way the notebook functions.

toolbar: The tool bar gives a quick way of performing the most-used operations within the notebook, by clicking on an icon.

code cell: the default type of cell, read on for an explanation of cells

---

Note: As of notebook version 4.1, the user interface allows for multiple cells to be selected. The `quick celltype selector`, found in the menubar, will display a dash – when multiple cells are selected to indicate that the type of the cells in the selection might not be unique. The quick selector can still be used to change the type of the selection and will change the type of all the currently selected cells.

1.4 Structure of a notebook document

The notebook consists of a sequence of cells. A cell is a multiline text input field, and its contents can be executed by using `Shift-Enter`, or by clicking either the "Play" button the toolbar, or *Cell | Run* in the menu bar. The execution behavior of a cell is determined the cell's type. There are four types of cells: **code cells**, **markdown cells**, **raw cells** and **heading cells**. Every cell starts off being a **code cell**, but its type can be changed by using a drop-down on the toolbar (which will be "Code", initially), or via *keyboard shortcuts*.

For more information on the different things you can do in a notebook, see the collection of examples.

1.4.1 Code cells

A *code cell* allows you to edit and write new code, with full syntax highlighting and tab completion. By default, the language associated to a code cell is Python, but other languages, such as `Julia` and `R`, can be handled using cell magic commands.

When a code cell is executed, code that it contains is sent to the kernel associated with the notebook. The results that are returned from this computation are then displayed in the notebook as the cell's *output*. The output is not limited to text, with many other possible forms of output are also possible, including `matplotlib` figures and HTML tables (as used, for example, in the `pandas` data analysis package). This is known as IPython's *rich display* capability.

See also:

Rich Output example notebook

1.4.2 Markdown cells

You can document the computational process in a literate way, alternating descriptive text with code, using *rich text*. In IPython this is accomplished by marking up text with the Markdown language. The corresponding cells are called *Markdown cells*. The Markdown language provides a simple way to perform this text markup, that is, to specify which parts of the text should be emphasized (italics), bold, form lists, etc.

When a Markdown cell is executed, the Markdown code is converted into the corresponding formatted rich text. Markdown allows arbitrary HTML code for formatting.

Within Markdown cells, you can also include *mathematics* in a straightforward way, using standard LaTeX notation: `$...$` for inline mathematics and `$$...$$` for displayed mathematics. When the Markdown cell is executed, the LaTeX portions are automatically rendered in the HTML output as equations with high quality typography. This is made possible by MathJax, which supports a large subset of LaTeX functionality

Standard mathematics environments defined by LaTeX and AMS-LaTeX (the *amsmath* package) also work, such as `\begin{equation}...\end{equation}`, and `\begin{align}...\end{align}`. New LaTeX macros may be defined using standard methods, such as `\newcommand`, by placing them anywhere *between math delimiters* in a Markdown cell. These definitions are then available throughout the rest of the IPython session.

See also:

Markdown Cells example notebook

1.4.3 Raw cells

Raw cells provide a place in which you can write *output* directly. Raw cells are not evaluated by the notebook. When passed through nbconvert, raw cells arrive in the destination format unmodified. For example, this allows you to type full LaTeX into a raw cell, which will only be rendered by LaTeX after conversion by nbconvert.

1.4.4 Heading cells

If you want to provide structure for your document, you can use markdown headings. Markdown headings consist of 1 to 6 hash # signs # followed by a space and the title of your section. The markdown heading will be converted to a clickable link for a section of the notebook. It is also used as a hint when exporting to other document formats, like PDF. We recommend using only one markdown header in a cell and limit the cell's content to the header text. For flexibility of text format conversion, we suggest placing additional text in the next notebook cell.

1.5 Basic workflow

The normal workflow in a notebook is, then, quite similar to a standard IPython session, with the difference that you can edit cells in-place multiple times until you obtain the desired results, rather than having to rerun separate scripts with the `%run` magic command.

Typically, you will work on a computational problem in pieces, organizing related ideas into cells and moving forward once previous parts work correctly. This is much more convenient for interactive exploration than breaking up a computation into scripts that must be executed together, as was previously necessary, especially if parts of them take a long time to run.

At certain moments, it may be necessary to interrupt a calculation which is taking too long to complete. This may be done with the *Kernel | Interrupt* menu option, or the `Ctrl-m i` keyboard shortcut. Similarly, it may be necessary or desirable to restart the whole computational process, with the *Kernel | Restart* menu option or `Ctrl-m .` shortcut.

A notebook may be downloaded in either a `.ipynb` or `.py` file from the menu option *File | Download as*. Choosing the `.py` option downloads a Python `.py` script, in which all rich output has been removed and the content of markdown cells have been inserted as comments.

See also:

Running Code in the Jupyter Notebook example notebook

Notebook Basics example notebook

a warning about doing "roundtrip" conversions.

1.5.1 Keyboard shortcuts

All actions in the notebook can be performed with the mouse, but keyboard shortcuts are also available for the most common ones. The essential shortcuts to remember are the following:

- **Shift-Enter: run cell** Execute the current cell, show output (if any), and jump to the next cell below. If `Shift-Enter` is invoked on the last cell, a new code cell will also be created. Note that in the notebook, typing `Enter` on its own *never* forces execution, but rather just inserts a new line in the current cell. `Shift-Enter` is equivalent to clicking the `Cell | Run` menu item.

- **Ctrl-Enter: run cell in-place** Execute the current cell as if it were in "terminal mode", where any output is shown, but the cursor *remains* in the current cell. The cell's entire contents are selected after execution, so you can just start typing and only the new input will be in the cell. This is convenient for doing quick experiments in place, or for querying things like filesystem content, without needing to create additional cells that you may not want to be saved in the notebook.

- **Alt-Enter: run cell, insert below** Executes the current cell, shows the output, and inserts a *new* cell between the current cell and the cell below (if one exists). This is thus a shortcut for the sequence `Shift-Enter`, `Ctrl-m a`. (`Ctrl-m a` adds a new cell above the current one.)

- **Esc and Enter: Command mode and edit mode** In command mode, you can easily navigate around the notebook using keyboard shortcuts. In edit mode, you can edit text in cells.

For the full list of available shortcuts, click *Help*, *Keyboard Shortcuts* in the notebook menus.

1.6 Plotting

One major feature of the Jupyter notebook is the ability to display plots that are the output of running code cells. The IPython kernel is designed to work seamlessly with the matplotlib plotting library to provide this functionality. Specific plotting library integration is a feature of the kernel.

1.7 Installing kernels

For information on how to install a Python kernel, refer to the IPython install page.

Kernels for other languages can be found in the IPython wiki. They usually come with instruction what to run to make the kernel available in the notebook.

1.8 Signing Notebooks

To prevent untrusted code from executing on users' behalf when notebooks open, we have added a signature to the notebook, stored in metadata. The notebook server verifies this signature when a notebook is opened. If the signature stored in the notebook metadata does not match, javascript and HTML output will not be displayed on load, and must be regenerated by re-executing the cells.

Any notebook that you have executed yourself *in its entirety* will be considered trusted, and its HTML and javascript output will be displayed on load.

If you need to see HTML or Javascript output without re-executing, you can explicitly trust notebooks, such as those shared with you, or those that you have written yourself prior to IPython 2.0, at the command-line with:

```
$ jupyter trust mynotebook.ipynb [other notebooks.ipynb]
```

This just generates a new signature stored in each notebook.

You can generate a new notebook signing key with:

```
$ jupyter trust --reset
```

1.9 Browser Compatibility

The Jupyter Notebook is officially supported the latest stable version the following browsers:

- Chrome
- Safari
- Firefox

The is mainly due to the notebook's usage of WebSockets and the flexible box model.

The following browsers are unsupported:

- Safari < 5
- Firefox < 6
- Chrome < 13
- Opera (any): CSS issues, but execution might work
- Internet Explorer < 10
- Internet Explorer 10 (same as Opera)

Using Safari with HTTPS and an untrusted certificate is known to not work (websockets will fail).

UI Components

When opening bug reports or sending emails to the Jupyter mailing list, it is useful to know the names of different UI components so that other developers and users have an easier time helping you diagnose your problems. This section will familiarize you with the names of UI elements within the Notebook and the different Notebook modes.
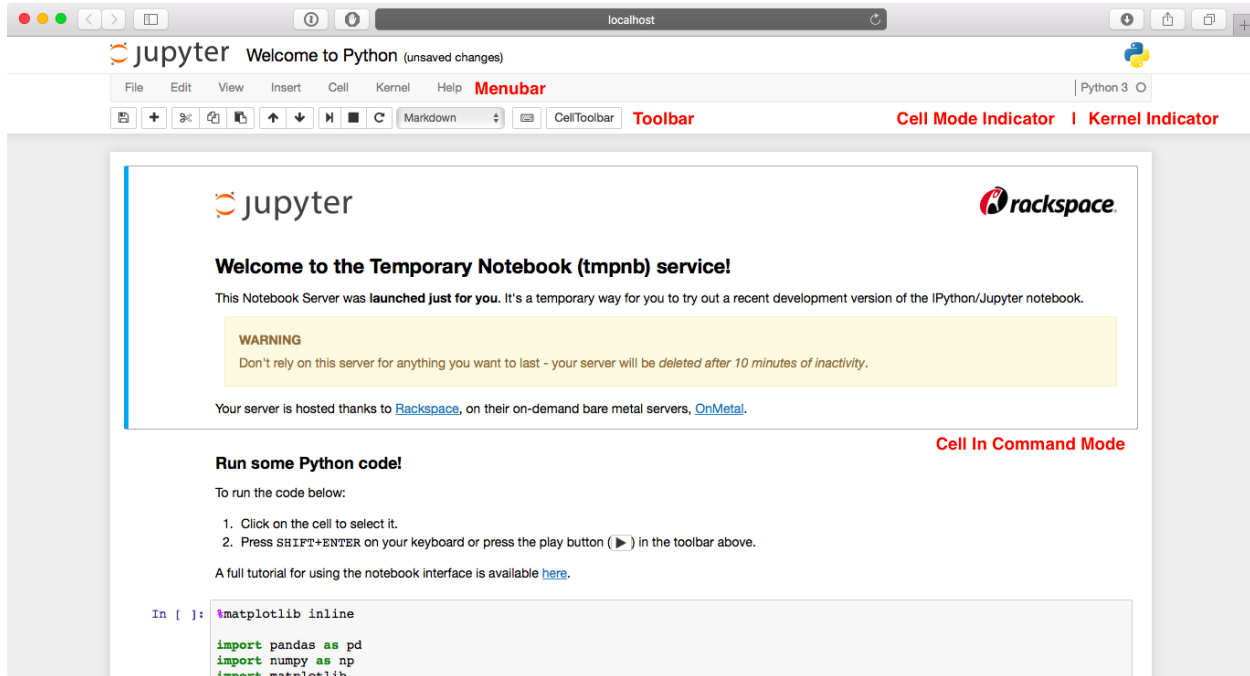
2.1 Notebook Dashboard

When you launch `jupyter notebook` the first page that you encounter is the Notebook Dashboard.
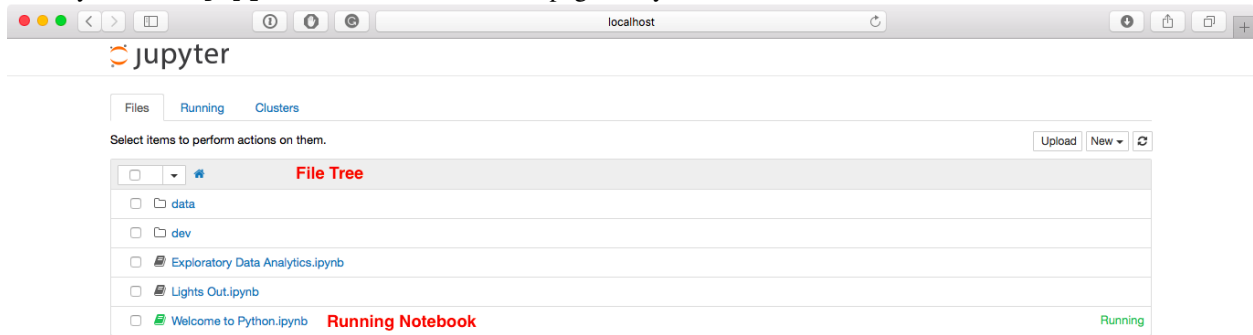
2.2 Notebook Editor

Once you've selected a Notebook to edit, the Notebook will open in the Notebook Editor.

2.3 Interactive User Interface Tour of the Notebook

If you would like to learn more about the specific elements within the Notebook Editor, you can go through the User Interface Tour by selecting Help in the menubar then selecting User Interface Tour.

2.3.1 Edit Mode and Notebook Editor

When a cell is in edit mode, the Cell Mode Indicator will change to reflect the cell's state. This state is indicated by a small pencil icon on the top right of the interface. When the cell is in command mode, there is no icon in that location.
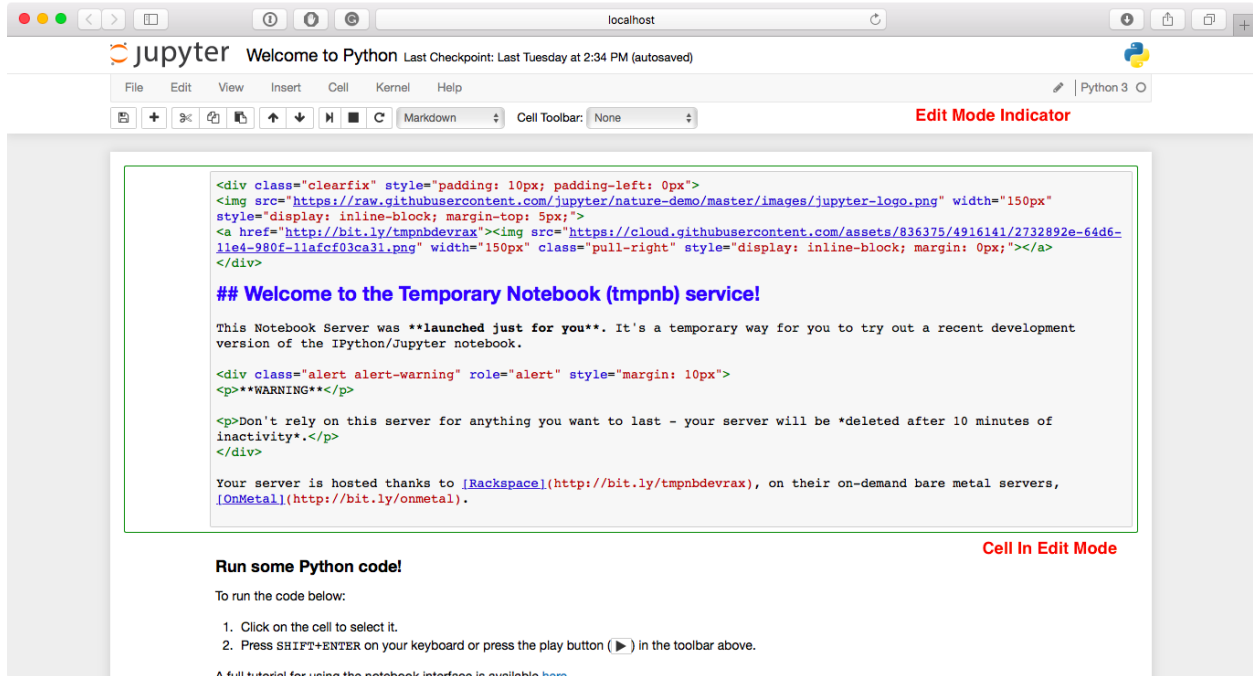
2.4 File Editor

Now let's say that you've chosen to open a Markdown file instead of a Notebook file whilst in the Notebook Dashboard. If so, the file will be opened in the File Editor.
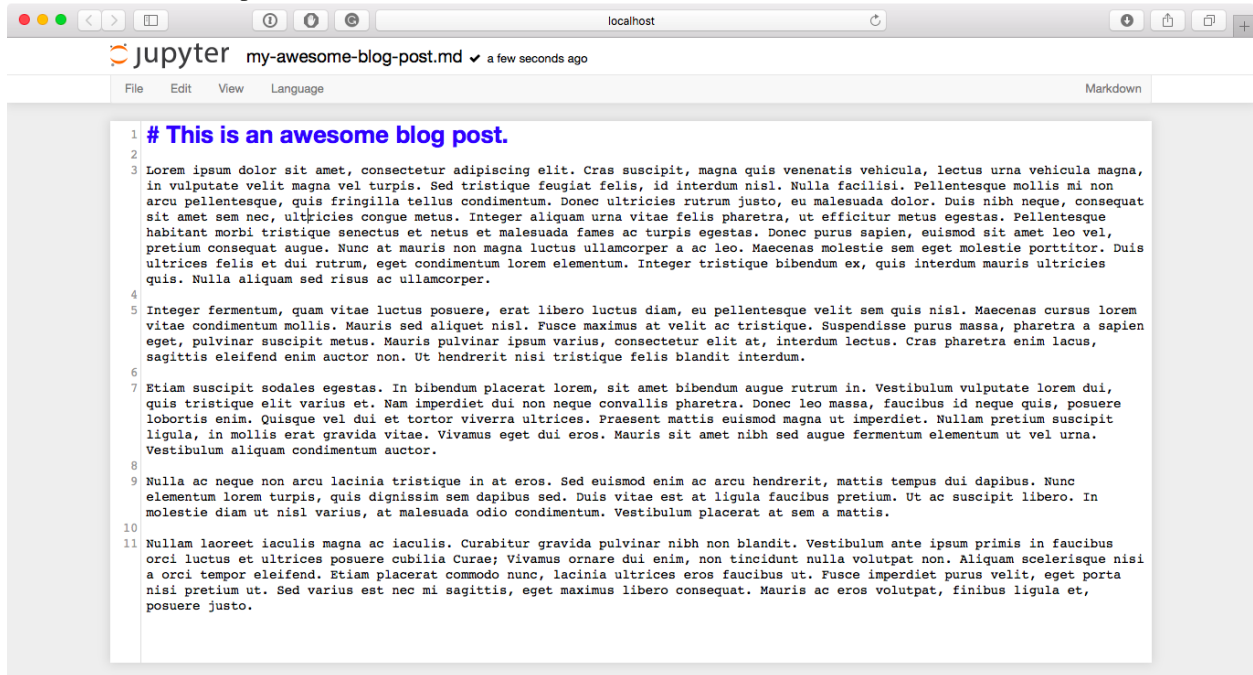
Configuration Overview

Beyond the default configuration settings, you can configure a rich array of options to suit your workflow. Here are areas that are commonly configured when using Jupyter Notebook:

- *Jupyter's common configuration system*
- *Notebook server*
- *Notebook front-end client*
- *Notebook extensions*

Let's look at highlights of each area.

3.1 Jupyter's Common Configuration system

Jupyter applications, from the Notebook to JupyterHub to nbgrader, share a common configuration system. The process for creating a configuration file and editing settings is similar for all the Jupyter applications.

- Configuring a Jupyter application
- Using Python to set up the configuration files
- Configuring a language kernel
- traitlets provide a low-level architecture for configuration.

3.2 Notebook server

The Notebook server runs the language kernel and communicates with the front-end Notebook client (i.e. the familiar notebook interface).

- Configuring the Notebook server

    To create a `jupyter_notebook_config.py` file in the `.jupyter` directory, with all the defaults commented out, use the following command:

    ```
    $ jupyter notebook --generate-config
    ```

    Command line arguments for configuration settings are documented in the configuration file and the user documentation.

- *Running a Notebook server*

- Related: Configuring a language kernel to run in the Notebook server enables your server to run other languages, like R or Julia.

3.3 Notebook front-end client

- *How front-end configuration works*
    - *Example: Changing the notebook's default indentation setting*
    - *Example: Restoring the notebook's default indentation setting*
- *Persisting configuration settings*

3.4 Notebook extensions

- Distributing Jupyter Extensions as Python Packages

- Extending the Notebook

Security in Jupyter notebooks: Since security policies vary from organization to organization, we encourage you to consult with your security team on settings that would be best for your use cases. Our documentation offers some responsible security practices, and we recommend becoming familiar with the practices.

Config file and command line options

The notebook server can be run with a variety of command line arguments. A list of available options can be found below in the *options section*.

Defaults for these options can also be set by creating a file named `jupyter_notebook_config.py` in your Jupyter folder. The Jupyter folder is in your home directory, `~/.jupyter`.

To create a `jupyter_notebook_config.py` file, with all the defaults commented out, you can use the following command line:

```
$ jupyter notebook --generate-config
```

4.1 Options

This list of options can be generated by running the following and hitting enter:

```
$ jupyter notebook --help
```

Application.log_datefmt [Unicode] Default: `'%Y-%m-%d %H:%M:%S'`

The date format used by logging formatters for %(asctime)s

Application.log_format [Unicode] Default: `'[%(name)s]%(highlevel)s %(message)s'`

The Logging format template

Application.log_level [0|10|20|30|40|50|'DEBUG'|'INFO'|'WARN'|'ERROR'|'CRITICAL'] Default: `30`

Set the log level by value or name.

JupyterApp.answer_yes [Bool] Default: `False`

Answer yes to any prompts.

JupyterApp.config_file [Unicode] Default: `''`

Full path of a config file.

JupyterApp.config_file_name [Unicode] Default: `''`

Specify a config file to load.

JupyterApp.generate_config [Bool] Default: `False`

Generate default config file.

NotebookApp.allow_credentials [Bool] Default: `False`

>   Set the Access-Control-Allow-Credentials: true header

NotebookApp.allow_origin [Unicode] Default: `''`

>   Set the Access-Control-Allow-Origin header

>   Use '*' to allow any origin to access your server.

>   Takes precedence over allow_origin_pat.

NotebookApp.allow_origin_pat [Unicode] Default: `''`

>   Use a regular expression for the Access-Control-Allow-Origin header

>   Requests from an origin matching the expression will get replies with:

>>   Access-Control-Allow-Origin: origin

>   where *origin* is the origin of the request.

>   Ignored if allow_origin is set.

NotebookApp.allow_root [Bool] Default: `False`

>   Whether to allow the user to run the notebook as root.

NotebookApp.base_project_url [Unicode] Default: `'/'`

>   DEPRECATED use base_url

NotebookApp.base_url [Unicode] Default: `'/'`

>   The base URL for the notebook server.

>   Leading and trailing slashes can be omitted, and will automatically be added.

NotebookApp.browser [Unicode] Default: `''`

>   Specify what command to use to invoke a web browser when opening the notebook. If not specified, the default browser will be determined by the *webbrowser* standard library module, which allows setting of the BROWSER environment variable to override it.

NotebookApp.certfile [Unicode] Default: `''`

>   The full path to an SSL/TLS certificate file.

NotebookApp.client_ca [Unicode] Default: `''`

>   The full path to a certificate authority certifificate for SSL/TLS client authentication.

NotebookApp.config_manager_class [Type] Default: `'notebook.services.config.manager.ConfigManager'`

>   The config manager class to use

NotebookApp.contents_manager_class [Type] Default: `'notebook.services.contents.filemanager.FileContents`

>   The notebook manager class to use.

NotebookApp.cookie_options [Dict] Default: `{}`

>   Extra keyword arguments to pass to *set_secure_cookie*. See tornado's set_secure_cookie docs for details.

NotebookApp.cookie_secret [Bytes] Default: `b''`

>   The random bytes used to secure cookies. By default this is a new random number every time you start the Notebook. Set it to a value in a config file to enable logins to persist across server sessions.

Note: Cookie secrets should be kept private, do not share config files with cookie_secret stored in plaintext (you can read the value from a file).

NotebookApp.cookie_secret_file [Unicode] Default: `''`

The file where the cookie secret is stored.

NotebookApp.default_url [Unicode] Default: `'/tree'`

The default URL to redirect to from `/`

NotebookApp.enable_mathjax [Bool] Default: `True`

Whether to enable MathJax for typesetting math/TeX

MathJax is the javascript library Jupyter uses to render math/LaTeX. It is very large, so you may want to disable it if you have a slow internet connection, or for offline use of the notebook.

When disabled, equations etc. will appear as their untransformed TeX source.

NotebookApp.extra_nbextensions_path [List] Default: `[]`

extra paths to look for Javascript notebook extensions

NotebookApp.extra_static_paths [List] Default: `[]`

Extra paths to search for serving static files.

This allows adding javascript/css to be available from the notebook server machine, or overriding individual files in the IPython

NotebookApp.extra_template_paths [List] Default: `[]`

Extra paths to search for serving jinja templates.

Can be used to override templates from notebook.templates.

NotebookApp.file_to_run [Unicode] Default: `''`

No description

NotebookApp.ignore_minified_js [Bool] Default: `False`

Deprecated: Use minified JS file or not, mainly use during dev to avoid JS recompilation

NotebookApp.iopub_data_rate_limit [Float] Default: `0`

(bytes/sec) Maximum rate at which messages can be sent on iopub before they are limited.

NotebookApp.iopub_msg_rate_limit [Float] Default: `0`

(msg/sec) Maximum rate at which messages can be sent on iopub before they are limited.

NotebookApp.ip [Unicode] Default: `'localhost'`

The IP address the notebook server will listen on.

NotebookApp.jinja_environment_options [Dict] Default: `{}`

Supply extra arguments that will be passed to Jinja environment.

NotebookApp.jinja_template_vars [Dict] Default: `{}`

Extra variables to supply to jinja templates when rendering.

NotebookApp.kernel_manager_class [Type] Default: `'notebook.services.kernels.kernelmanager.MappingKerne`

The kernel manager class to use.

NotebookApp.kernel_spec_manager_class [Type] Default: `'jupyter_client.kernelspec.KernelSpecManager'`

The kernel spec manager class to use. Should be a subclass of *jupyter_client.kernelspec.KernelSpecManager*.

The Api of KernelSpecManager is provisional and might change without warning between this version of Jupyter and the next stable one.

NotebookApp.keyfile [Unicode] Default: `''`

The full path to a private key file for usage with SSL/TLS.

NotebookApp.login_handler_class [Type] Default: `'notebook.auth.login.LoginHandler'`

The login handler class to use.

NotebookApp.logout_handler_class [Type] Default: `'notebook.auth.logout.LogoutHandler'`

The logout handler class to use.

NotebookApp.mathjax_url [Unicode] Default: `''`

The url for MathJax.js.

NotebookApp.nbserver_extensions [Dict] Default: `{}`

Dict of Python modules to load as notebook server extensions.Entry values can be used to enable and disable the loading ofthe extensions.

NotebookApp.notebook_dir [Unicode] Default: `''`

The directory to use for notebooks and kernels.

NotebookApp.open_browser [Bool] Default: `True`

Whether to open in a browser after starting. The specific browser used is platform dependent and determined by the python standard library *webbrowser* module, unless it is overridden using the –browser (NotebookApp.browser) configuration option.

NotebookApp.password [Unicode] Default: `''`

Hashed password to use for web authentication.

To generate, type in a python/IPython shell:

> from notebook.auth import passwd; passwd()

The string should be of the form type:salt:hashed-password.

NotebookApp.password_required [Bool] Default: `False`

Forces users to use a password for the Notebook server. This is useful in a multi user environment, for instance when everybody in the LAN can access each other's machine though ssh.

In such a case, server the notebook server on localhost is not secure since any user can connect to the notebook server via ssh.

NotebookApp.port [Int] Default: `8888`

The port the notebook server will listen on.

NotebookApp.port_retries [Int] Default: `50`

The number of additional ports to try if the specified port is not available.

NotebookApp.pylab [Unicode] Default: `'disabled'`

DISABLED: use %pylab or %matplotlib in the notebook to enable matplotlib.

NotebookApp.rate_limit_window [Float] Default: `1.0`

> (sec) Time window used to check the message and data rate limits.

NotebookApp.reraise_server_extension_failures [Bool] Default: `False`

> Reraise exceptions encountered loading server extensions?

NotebookApp.server_extensions [List] Default: `[]`

> DEPRECATED use the nbserver_extensions dict instead

NotebookApp.session_manager_class [Type] Default: `'notebook.services.sessions.sessionmanager.SessionManager`

> The session manager class to use.

NotebookApp.ssl_options [Dict] Default: `{}`

> Supply SSL options for the tornado HTTPServer. See the tornado docs for details.

NotebookApp.tornado_settings [Dict] Default: `{}`

> Supply overrides for the tornado.web.Application that the Jupyter notebook uses.

NotebookApp.trust_xheaders [Bool] Default: `False`

> Whether to trust or not X-Scheme/X-Forwarded-Proto and X-Real-Ip/X-Forwarded-For headerssent by the upstream reverse proxy. Necessary if the proxy handles SSL

NotebookApp.webapp_settings [Dict] Default: `{}`

> DEPRECATED, use tornado_settings

NotebookApp.websocket_url [Unicode] Default: `''`

> The base URL for websockets, if it differs from the HTTP server (hint: it almost certainly doesn't).
>
> Should be in the form of an HTTP origin: ws[s]://hostname[:port]

ConnectionFileMixin.connection_file [Unicode] Default: `''`

> JSON file in which to store connection info [default: kernel-<pid>.json]
>
> This file will contain the IP, ports, and authentication key needed to connect clients to this kernel. By default, this file will be created in the security dir of the current profile, but can be specified by absolute path.

ConnectionFileMixin.control_port [Int] Default: `0`

> set the control (ROUTER) port [default: random]

ConnectionFileMixin.hb_port [Int] Default: `0`

> set the heartbeat port [default: random]

ConnectionFileMixin.iopub_port [Int] Default: `0`

> set the iopub (PUB) port [default: random]

ConnectionFileMixin.ip [Unicode] Default: `''`

> Set the kernel's IP address [default localhost]. If the IP address is something other than localhost, then Consoles on other machines will be able to connect to the Kernel, so be careful!

ConnectionFileMixin.shell_port [Int] Default: `0`

> set the shell (ROUTER) port [default: random]

ConnectionFileMixin.stdin_port [Int] Default: `0`

> set the stdin (ROUTER) port [default: random]

ConnectionFileMixin.transport ['tcp'|'ipc'] Default: `'tcp'`

>   No description

KernelManager.autorestart [Bool] Default: `True`

>   Should we autorestart the kernel if it dies.

KernelManager.kernel_cmd [List] Default: `[]`

>   DEPRECATED: Use kernel_name instead.

>   The Popen Command to launch the kernel. Override this if you have a custom kernel. If kernel_cmd is specified in a configuration file, Jupyter does not pass any arguments to the kernel, because it cannot make any assumptions about the arguments that the kernel understands. In particular, this means that the kernel does not receive the option –debug if it given on the Jupyter command line.

Session.buffer_threshold [Int] Default: `1024`

>   Threshold (in bytes) beyond which an object's buffer should be extracted to avoid pickling.

Session.check_pid [Bool] Default: `True`

>   Whether to check PID to protect against calls after fork.

>   This check can be disabled if fork-safety is handled elsewhere.

Session.copy_threshold [Int] Default: `65536`

>   Threshold (in bytes) beyond which a buffer should be sent without copying.

Session.debug [Bool] Default: `False`

>   Debug output in the Session

Session.digest_history_size [Int] Default: `65536`

>   The maximum number of digests to remember.

>   The digest history will be culled when it exceeds this value.

Session.item_threshold [Int] Default: `64`

>   The maximum number of items for a container to be introspected for custom serialization. Containers larger than this are pickled outright.

Session.key [CBytes] Default: `b''`

>   execution key, for signing messages.

Session.keyfile [Unicode] Default: `''`

>   path to file containing execution key.

Session.metadata [Dict] Default: `{}`

>   Metadata dictionary, which serves as the default top-level metadata dict for each message.

Session.packer [DottedObjectName] Default: `'json'`

>   The name of the packer for serializing messages. Should be one of 'json', 'pickle', or an import name for a custom callable serializer.

Session.session [CUnicode] Default: `''`

>   The UUID identifying this session.

Session.signature_scheme [Unicode] Default: `'hmac-sha256'`

>   The digest scheme used to construct the message signatures. Must have the form 'hmac-HASH'.

Session.unpacker [DottedObjectName] Default: `'json'`

> The name of the unpacker for unserializing messages. Only used with custom functions for *packer*.

Session.username [Unicode] Default: `'username'`

> Username for the Session. Default is your system username.

MultiKernelManager.default_kernel_name [Unicode] Default: `'python3'`

> The name of the default kernel to start

MultiKernelManager.kernel_manager_class [DottedObjectName] Default: `'jupyter_client.ioloop.IOLoopKernelMan`

> The kernel manager class. This is configurable to allow subclassing of the KernelManager for customized behavior.

MappingKernelManager.root_dir [Unicode] Default: `''`

> No description

ContentsManager.checkpoints [Instance] Default: `None`

> No description

ContentsManager.checkpoints_class [Type] Default: `'notebook.services.contents.checkpoints.Checkpoints'`

> No description

ContentsManager.checkpoints_kwargs [Dict] Default: `{}`

> No description

ContentsManager.hide_globs [List] Default: `['__pycache__', '*.pyc', '*.pyo', '.DS_Store',`
`'*.so', '*.dyl...`

> Glob patterns to hide in file and directory listings.

ContentsManager.pre_save_hook [Any] Default: `None`

> Python callable or importstring thereof
>
> To be called on a contents model prior to save.
>
> This can be used to process the structure, such as removing notebook outputs or other side effects that should not be saved.
>
> It will be called as (all arguments passed by keyword):

```
hook(path=path, model=model, contents_manager=self)
```

> - model: the model to be saved. Includes file contents. Modifying this dict will affect the file that is stored.
> - path: the API path of the save destination
> - contents_manager: this ContentsManager instance

ContentsManager.untitled_directory [Unicode] Default: `'Untitled Folder'`

> The base name used when creating untitled directories.

ContentsManager.untitled_file [Unicode] Default: `'untitled'`

> The base name used when creating untitled files.

ContentsManager.untitled_notebook [Unicode] Default: `'Untitled'`

> The base name used when creating untitled notebooks.

FileManagerMixin.use_atomic_writing [Bool] Default: `True`

> By default notebooks are saved on disk on a temporary file and then if succefully written, it replaces the old ones. This procedure, namely 'atomic_writing', causes some bugs on file system whitout operation order enforcement (like some networked fs). If set to False, the new notebook is written directly on the old one which could fail (eg: full filesystem or quota)

FileContentsManager.post_save_hook [Any] Default: `None`

> Python callable or importstring thereof
>
> to be called on the path of a file just saved.
>
> This can be used to process the file on disk, such as converting the notebook to a script or HTML via nbconvert.
>
> It will be called as (all arguments passed by keyword):

```
hook(os_path=os_path, model=model, contents_manager=instance)
```

> - path: the filesystem path to the file just written
>
> - model: the model representing the file
>
> - contents_manager: this ContentsManager instance

FileContentsManager.root_dir [Unicode] Default: `''`

> No description

FileContentsManager.save_script [Bool] Default: `False`

> DEPRECATED, use post_save_hook. Will be removed in Notebook 5.0

NotebookNotary.algorithm ['sha256'|'sha224'|'sha384'|'sha1'|'sha512'|'md5'] Default: `'sha256'`

> The hashing algorithm used to sign notebooks.

NotebookNotary.cache_size [Int] Default: `65535`

> The number of notebook signatures to cache. When the number of signatures exceeds this value, the oldest 25% of signatures will be culled.

NotebookNotary.db_file [Unicode] Default: `''`

> The sqlite file in which to store notebook signatures. By default, this will be in your Jupyter runtime directory. You can set it to ':memory:' to disable sqlite writing to the filesystem.

NotebookNotary.secret [Bytes] Default: `b''`

> The secret key with which notebooks are signed.

NotebookNotary.secret_file [Unicode] Default: `''`

> The file where the secret key is stored.

KernelSpecManager.ensure_native_kernel [Bool] Default: `True`

> If there is no Python kernelspec registered and the IPython kernel is available, ensure it is added to the spec list.

KernelSpecManager.kernel_spec_class [Type] Default: `'jupyter_client.kernelspec.KernelSpec'`

> The kernel spec class. This is configurable to allow subclassing of the KernelSpecManager for customized behavior.

KernelSpecManager.whitelist [Set] Default: `set()`

> Whitelist of allowed kernel names.
>
> By default, all installed kernels are allowed.

Running a notebook server

The Jupyter notebook web application is based on a server-client structure. The notebook server uses a two-process kernel architecture based on ZeroMQ, as well as Tornado for serving HTTP requests.

Note: By default, a notebook server runs locally at 127.0.0.1:8888 and is accessible only from *localhost*. You may access the notebook server from the browser using *http://127.0.0.1:8888*.

This document describes how you can *secure a notebook server* and how to *run it on a public interface*.

5.1 Securing a notebook server

You can protect your notebook server with a simple single password by configuring the `NotebookApp.password` setting in `jupyter_notebook_config.py`.

5.1.1 Prerequisite: A notebook configuration file

Check to see if you have a notebook configuration file, `jupyter_notebook_config.py`. The default location for this file is your Jupyter folder in your home directory, `~/.jupyter`.

If you don't already have one, create a config file for the notebook using the following command:

```
$ jupyter notebook --generate-config
```

5.1.2 Preparing a hashed password

You can prepare a hashed password using the function `notebook.auth.security.passwd()`:

```
In [1]: from notebook.auth import passwd
In [2]: passwd()
Enter password:
Verify password:
Out[2]: 'sha1:67c9e60bb8b6:9ffede0825894254b2e042ea597d771089e11aed'
```

Caution: `passwd()` when called with no arguments will prompt you to enter and verify your password such as in the above code snippet. Although the function can also be passed a string as an argument such as `passwd('mypassword')`, please **do not** pass a string as an argument inside an IPython session, as it will be saved in your input history.

5.1.3 Adding hashed password to your notebook configuration file

You can then add the hashed password to your `jupyter_notebook_config.py`. The default location for this file `jupyter_notebook_config.py` is in your Jupyter folder in your home directory, `~/.jupyter`, e.g.:

```
c.NotebookApp.password = u'sha1:67c9e60bb8b6:9ffede0825894254b2e042ea597d771089e11aed'
```

5.1.4 Using SSL for encrypted communication

When using a password, it is a good idea to also use SSL with a web certificate, so that your hashed password is not sent unencrypted by your browser.

Important: Web security is rapidly changing and evolving. We provide this document as a convenience to the user, and recommend that the user keep current on changes that may impact security, such as new releases of OpenSSL. The Open Web Application Security Project (OWASP) website is a good resource on general security issues and web practices.

You can start the notebook to communicate via a secure protocol mode by setting the `certfile` option to your self-signed certificate, i.e. `mycert.pem`, with the command:

```
$ jupyter notebook --certfile=mycert.pem --keyfile mykey.key
```

Tip: A self-signed certificate can be generated with `openssl`. For example, the following command will create a certificate valid for 365 days with both the key and certificate data written to the same file:

```
$ openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout mykey.key -out mycert.pem
```

When starting the notebook server, your browser may warn that your self-signed certificate is insecure or unrecognized. If you wish to have a fully compliant self-signed certificate that will not raise warnings, it is possible (but rather involved) to create one, as explained in detail in this tutorial.

5.2 Running a public notebook server

If you want to access your notebook server remotely via a web browser, you can do so by running a public notebook server. For optimal security when running a public notebook server, you should first secure the server with a password and SSL/HTTPS as described in *Securing a notebook server*.

Start by creating a certificate file and a hashed password, as explained in *Securing a notebook server*.

If you don't already have one, create a config file for the notebook using the following command line:

```
$ jupyter notebook --generate-config
```

In the `~/.jupyter` directory, edit the notebook config file, `jupyter_notebook_config.py`. By default, the notebook config file has all fields commented out. The minimum set of configuration options that you should to uncomment and edit in :file:`jupyter_notebook_config.py` is the following:

```
# Set options for certfile, ip, password, and toggle off browser auto-opening
c.NotebookApp.certfile = u'/absolute/path/to/your/certificate/mycert.pem'
c.NotebookApp.keyfile = u'/absolute/path/to/your/certificate/mykey.key'
# Set ip to '*' to bind on all interfaces (ips) for the public server
c.NotebookApp.ip = '*'
```

```
c.NotebookApp.password = u'sha1:bcd259ccf...<your hashed password here>'
c.NotebookApp.open_browser = False

# It is a good idea to set a known, fixed port for server access
c.NotebookApp.port = 9999
```

You can then start the notebook using the `jupyter notebook` command.

Important: Use 'https'. Keep in mind that when you enable SSL support, you must access the notebook server over `https://`, not over plain `http://`. The startup message from the server prints a reminder in the console, but *it is easy to overlook this detail and think the server is for some reason non-responsive.*

When using SSL, always access the notebook server with 'https://'.

You may now access the public server by pointing your browser to `https://your.host.com:9999` where `your.host.com` is your public server's domain.

5.2.1 Firewall Setup

To function correctly, the firewall on the computer running the jupyter notebook server must be configured to allow connections from client machines on the access port `c.NotebookApp.port` set in :file:`jupyter_notebook_config.py` port to allow connections to the web interface. The firewall must also allow connections from 127.0.0.1 (localhost) on ports from 49152 to 65535. These ports are used by the server to communicate with the notebook kernels. The kernel communication ports are chosen randomly by ZeroMQ, and may require multiple connections per kernel, so a large range of ports must be accessible.

5.3 Running the notebook with a customized URL prefix

The notebook dashboard, which is the landing page with an overview of the notebooks in your working directory, is typically found and accessed at the default URL `http://localhost:8888/`.

If you prefer to customize the URL prefix for the notebook dashboard, you can do so through modifying `jupyter_notebook_config.py`. For example, if you prefer that the notebook dashboard be located with a sub-directory that contains other ipython files, e.g. `http://localhost:8888/ipython/`, you can do so with configuration options like the following (see above for instructions about modifying `jupyter_notebook_config.py`):

```
c.NotebookApp.base_url = '/ipython/'
```

5.4 Embedding the notebook in another website

Sometimes you may want to embed the notebook somewhere on your website, e.g. in an IFrame. To do this, you may need to override the Content-Security-Policy to allow embedding. Assuming your website is at *https://mywebsite.example.com*, you can embed the notebook on your website with the following configuration setting in `jupyter_notebook_config.py`:

```
c.NotebookApp.tornado_settings = {
    'headers': {
        'Content-Security-Policy': "frame-ancestors 'https://mywebsite.example.com' 'self' "
    }
}
```

When embedding the notebook in a website using an iframe, consider putting the notebook in single-tab mode. Since the notebook opens some links in new tabs by default, single-tab mode keeps the notebook from opening additional tabs. Adding the following to ~/.jupyter/custom/custom.js will enable single-tab mode:

```
define(['base/js/namespace'], function(Jupyter){
    Jupyter._target = '_self';
});
```

5.5 Known issues

5.5.1 Proxies

When behind a proxy, especially if your system or browser is set to autodetect the proxy, the notebook web application might fail to connect to the server's websockets, and present you with a warning at startup. In this case, you need to configure your system not to use the proxy for the server's address.

For example, in Firefox, go to the Preferences panel, Advanced section, Network tab, click 'Settings...', and add the address of the notebook server to the 'No proxy for' field.

5.5.2 Docker CMD

Using jupyter notebook as a Docker CMD results in kernels repeatedly crashing, likely due to a lack of PID reaping. To avoid this, use the tini init as your Dockerfile *ENTRYPOINT*:

```
# Add Tini. Tini operates as a process subreaper for jupyter. This prevents
# kernel crashes.
ENV TINI_VERSION v0.6.0
ADD https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini /usr/bin/tini
RUN chmod +x /usr/bin/tini
ENTRYPOINT ["/usr/bin/tini", "--"]

EXPOSE 8888
CMD ["jupyter", "notebook", "--port=8888", "--no-browser", "--ip=0.0.0.0"]
```

Security in Jupyter notebooks

As Jupyter notebooks become more popular for sharing and collaboration, the potential for malicious people to attempt to exploit the notebook for their nefarious purposes increases. IPython 2.0 introduces a security model to prevent execution of untrusted code without explicit user input.

6.1 The problem

The whole point of Jupyter is arbitrary code execution. We have no desire to limit what can be done with a notebook, which would negatively impact its utility.

Unlike other programs, a Jupyter notebook document includes output. Unlike other documents, that output exists in a context that can execute code (via Javascript).

The security problem we need to solve is that no code should execute just because a user has **opened** a notebook that **they did not write**. Like any other program, once a user decides to execute code in a notebook, it is considered trusted, and should be allowed to do anything.

6.2 Our security model

- Untrusted HTML is always sanitized

- Untrusted Javascript is never executed

- HTML and Javascript in Markdown cells are never trusted

- **Outputs** generated by the user are trusted

- Any other HTML or Javascript (in Markdown cells, output generated by others) is never trusted

- The central question of trust is "Did the current user do this?"

6.3 The details of trust

Jupyter notebooks store a signature in metadata, which is used to answer the question "Did the current user do this?"

This signature is a digest of the notebooks contents plus a secret key, known only to the user. The secret key is a user-only readable file in the Jupyter profile's security directory. By default, this is:

```
~/.jupyter/profile_default/security/notebook_secret
```

Note: The notebook secret being stored in the profile means that loading a notebook in another profile results in it being untrusted, unless you copy or symlink the notebook secret to share it across profiles.

When a notebook is opened by a user, the server computes a signature with the user's key, and compares it with the signature stored in the notebook's metadata. If the signature matches, HTML and Javascript output in the notebook will be trusted at load, otherwise it will be untrusted.

Any output generated during an interactive session is trusted.

6.3.1 Updating trust

A notebook's trust is updated when the notebook is saved. If there are any untrusted outputs still in the notebook, the notebook will not be trusted, and no signature will be stored. If all untrusted outputs have been removed (either via `Clear Output` or re-execution), then the notebook will become trusted.

While trust is updated per output, this is only for the duration of a single session. A notebook file on disk is either trusted or not in its entirety.

6.3.2 Explicit trust

Sometimes re-executing a notebook to generate trusted output is not an option, either because dependencies are unavailable, or it would take a long time. Users can explicitly trust a notebook in two ways:

- At the command-line, with:

```
jupyter trust /path/to/notebook.ipynb
```

- After loading the untrusted notebook, with `File / Trust Notebook`

These two methods simply load the notebook, compute a new signature with the user's key, and then store the newly signed notebook.

6.4 Reporting security issues

If you find a security vulnerability in Jupyter, either a failure of the code to properly implement the model described here, or a failure of the model itself, please report it to security@ipython.org.

If you prefer to encrypt your security reports, you can use `this PGP public key`.

6.5 Affected use cases

Some use cases that work in Jupyter 1.0 will become less convenient in 2.0 as a result of the security changes. We do our best to minimize these annoyance, but security is always at odds with convenience.

6.5.1 Javascript and CSS in Markdown cells

While never officially supported, it had become common practice to put hidden Javascript or CSS styling in Markdown cells, so that they would not be visible on the page. Since Markdown cells are now sanitized (by Google Caja), all Javascript (including click event handlers, etc.) and CSS will be stripped.

We plan to provide a mechanism for notebook themes, but in the meantime styling the notebook can only be done via either `custom.css` or CSS in HTML output. The latter only have an effect if the notebook is trusted, because otherwise the output will be sanitized just like Markdown.

6.5.2 Collaboration

When collaborating on a notebook, people probably want to see the outputs produced by their colleagues' most recent executions. Since each collaborator's key will differ, this will result in each share starting in an untrusted state. There are three basic approaches to this:

- re-run notebooks when you get them (not always viable)

- explicitly trust notebooks via `jupyter trust` or the notebook menu (annoying, but easy)

- share a notebook secret, and use a Jupyter profile dedicated to the collaboration while working on the project.

6.5.3 Multiple profiles or machines

Since the notebook secret is stored in a profile directory by default, opening a notebook with a different profile or on a different machine will result in a different key, and thus be untrusted. The only current way to address this is by sharing the notebook secret. This can be facilitated by setting the configurable:

```
c.NotebookApp.secret_file = "/path/to/notebook_secret"
```

in each profile, and only sharing the secret once per machine.

Configuring the notebook frontend

Note: The ability to configure the notebook frontend UI and preferences is still a work in progress.

This document is a rough explanation on how you can persist some configuration options for the notebook JavaScript.

There is no exhaustive list of all the configuration options as most options are passed down to other libraries, which means that non valid configuration can be ignored without any error messages.

7.1 How front end configuration works

The frontend configuration system works as follows:

- get a handle of a configurable JavaScript object.
- access its configuration attribute.
- update its configuration attribute with a JSON patch.

7.2 Example - Changing the notebook's default indentation

This example explains how to change the default setting `indentUnit` for CodeMirror Code Cells:

```
var cell = Jupyter.notebook.get_selected_cell();
var config = cell.config;
var patch = {
      CodeCell:{
        cm_config:{indentUnit:2}
      }
    }
config.update(patch)
```

You can enter the previous snippet in your browser's JavaScript console once. Then reload the notebook page in your browser. Now, the preferred indent unit should be equal to two spaces. The custom setting persists and you do not need to reissue the patch on new notebooks.

`indentUnit`, used in this example, is one of the many CodeMirror options which are available for configuration.

7.3 Example - Restoring the notebook's default indentation

If you want to restore a notebook frontend preference to its default value, you will enter a JSON patch with a `null` value for the preference setting.

For example, let's restore the indent setting `indentUnit` to its default of four spaces. Enter the following code snippet in your JavaScript console:

```
var cell = Jupyter.notebook.get_selected_cell();
var config = cell.config;
var patch = {
    CodeCell:{
      cm_config:{indentUnit: null} # only change here.
    }
  }
config.update(patch)
```

Reload the notebook in your browser and the default indent should again be two spaces.

7.4 Persisting configuration settings

Under the hood, Jupyter will persist the preferred configuration settings in `~/.jupyter/nbconfig/<section>.json`, with `<section>` taking various value depending on the page where the configuration is issued. `<section>` can take various values like `notebook`, `tree`, and `editor`. A `common` section contains configuration settings shared by all pages.

Distributing Jupyter Extensions as Python Packages

8.1 Overview

8.1.1 How can the notebook be extended?

The Jupyter Notebook client and server application are both deeply customizable. Their behavior can be extended by creating, respectively:

- nbextension: a notebook extension
  - a single JS file, or directory of JavaScript, Cascading StyleSheets, etc. that contain at minimum a JavaScript module packaged as an AMD modules that exports a function `load_ipython_extension`
- server extension: an importable Python module
  - that implements `load_jupyter_server_extension`

8.1.2 Why create a Python package for Jupyter extensions?

Since it is rare to have a server extension that does not have any frontend components (an nbextension), for convenience and consistency, all these client and server extensions with their assets can be packaged and versioned together as a Python package with a few simple commands. This makes installing the package of extensions easier and less error-prone for the user.

8.2 Installation of Jupyter Extensions

8.2.1 Install a Python package containing Jupyter Extensions

There are several ways that you may get a Python package containing Jupyter Extensions. Commonly, you will use a package manager for your system:

```
pip install helpful_package
# or
conda install helpful_package
# or
apt-get install helpful_package

# where 'helpful_package' is a Python package containing one or more Jupyter Extensions
```

8.2.2 Enable a Server Extension

The simplest case would be to enable a server extension which has no frontend components.

A `pip` user that wants their configuration stored in their home directory would type the following command:

```
jupyter serverextension enable --py helpful_package
```

Alternatively, a `virtualenv` or `conda` user can pass `--sys-prefix` which keeps their environment isolated and reproducible. For example:

```
# Make sure that your virtualenv or conda environment is activated
[source] activate my-environment

jupyter serverextension enable --py helpful_package --sys-prefix
```

8.2.3 Install the nbextension assets

If a package also has an nbextension with frontend assets that must be available (but not neccessarily enabled by default), install these assets with the following command:

```
jupyter nbextension install --py helpful_package # or --sys-prefix if using virtualenv or conda
```

8.2.4 Enable nbextension assets

If a package has assets that should be loaded every time a Jupyter app (e.g. lab, notebook, dashboard, terminal) is loaded in the browser, the following command can be used to enable the nbextension:

```
jupyter nbextension enable --py helpful_package # or --sys-prefix if using virtualenv or conda
```

8.3 Did it work? Check by listing Jupyter Extensions.

After running one or more extension installation steps, you can list what is presently known about nbextensions or server extension. The following commands will list which extensions are available, whether they are enabled, and other extension details:

```
jupyter nbextension list
jupyter serverextension list
```

8.4 Additional resources on creating and distributing packages

Of course, in addition to the files listed, there are number of other files one needs to build a proper package. Here are some good resources: - The Hitchhiker's Guide to Packaging - Repository Structure and Python by Kenneth Reitz

How you distribute them, too, is important: - Packaging and Distributing Projects - conda: Building packages

Here are some tools to get you started: - generator-nbextension

8.5 Example - Server extension

8.5.1 Creating a Python package with a server extension

Here is an example of a python module which contains a server extension directly on itself. It has this directory structure:

```
- setup.py
- MANIFEST.in
- my_module/
  - __init__.py
```

8.5.2 Defining the server extension

This example shows that the server extension and its `load_jupyter_server_extension` function are defined in the `__init__.py` file. #### my_module/__init__.py

```python
def _jupyter_server_extension_paths():
    return [{
        "module": "my_module"
    }]


def load_jupyter_server_extension(nbapp):
    nbapp.log.info("my module enabled!")
```

8.5.3 Install and enable the server extension

Which a user can install with:

```
jupyter serverextension enable --py my_module [--sys-prefix]
```

8.6 Example - Server extension and nbextension

8.6.1 Creating a Python package with a server extension and nbextension

Here is another server extension, with a front-end module. It assumes this directory structure:

```
- setup.py
- MANIFEST.in
- my_fancy_module/
  - __init__.py
  - static/
    index.js
```

8.6.2 Defining the server extension and nbextension

This example again shows that the server extension and its `load_jupyter_server_extension` function are defined in the `__init__.py` file. This time, there is also a function _jupyter_nbextension_path for the nbextension.

my_fancy_module/__init__.py

```python
def _jupyter_server_extension_paths():
    return [{
        "module": "my_fancy_module"
    }]

# Jupyter Extension points
def _jupyter_nbextension_paths():
    return [dict(
        section="notebook",
        # the path is relative to the `my_fancy_module` directory
        src="static",
        # directory in the `nbextension/` namespace
        dest="my_fancy_module",
        # _also_ in the `nbextension/` namespace
        require="my_fancy_module/index")]

def load_jupyter_server_extension(nbapp):
    nbapp.log.info("my module enabled!")
```

8.6.3 Install and enable the server extension and nbextension

The user can install and enable the extensions with the following set of commands:

```
jupyter nbextension install --py my_fancy_module [--sys-prefix|--user]
jupyter nbextension enable --py my_fancy_module [--sys-prefix|--system]
jupyter serverextension enable --py my_fancy_module [--sys-prefix|--system]
```

Extending the Notebook

Certain subsystems of the notebook server are designed to be extended or overridden by users. These documents explain these systems, and show how to override the notebook's defaults with your own custom behavior.

9.1 Contents API

The Jupyter Notebook web application provides a graphical interface for creating, opening, renaming, and deleting files in a virtual filesystem.

The `ContentsManager` class defines an abstract API for translating these interactions into operations on a particular storage medium. The default implementation, `FileContentsManager`, uses the local filesystem of the server for storage and straightforwardly serializes notebooks into JSON. Users can override these behaviors by supplying custom subclasses of ContentsManager.

This section describes the interface implemented by ContentsManager subclasses. We refer to this interface as the **Contents API**.

9.1.1 Data Model

Filesystem Entities

ContentsManager methods represent virtual filesystem entities as dictionaries, which we refer to as **models**.

Models may contain the following entries:

Key	Type	Info
name	unicode	Basename of the entity.
path	unicode	Full (*API-style*) path to the entity.
type	unicode	The entity type. One of `"notebook"`, `"file"` or `"directory"`.
created	datetime	Creation date of the entity.
last_modified	datetime	Last modified date of the entity.
content	variable	The "content" of the entity. (*See Below*)
mimetype	unicode or `None`	The mimetype of `content`, if any. (*See Below*)
format	unicode or `None`	The format of `content`, if any. (*See Below*)

Certain model fields vary in structure depending on the `type` field of the model. There are three model types: **notebook**, **file**, and **directory** .

- **notebook models**

    - The `format` field is always `"json"`.

- The `mimetype` field is always `None`.

- The `content` field contains a `nbformat.notebooknode.NotebookNode` representing the .ipynb file represented by the model. See the NBFormat documentation for a full description.

- **`file` models**

  - The `format` field is either `"text"` or `"base64"`.

  - The `mimetype` field is `text/plain` for text-format models and `application/octet-stream` for base64-format models.

  - The `content` field is always of type `unicode`. For text-format file models, `content` simply contains the file's bytes after decoding as UTF-8. Non-text (`base64`) files are read as bytes, base64 encoded, and then decoded as UTF-8.

- **`directory` models**

  - The `format` field is always `"json"`.

  - The `mimetype` field is always `None`.

  - The `content` field contains a list of *content-free* models representing the entities in the directory.

Note: In certain circumstances, we don't need the full content of an entity to complete a Contents API request. In such cases, we omit the `mimetype`, `content`, and `format` keys from the model. This most commonly occurs when listing a directory, in which circumstance we represent files within the directory as content-less models to avoid having to recursively traverse and serialize the entire filesystem.

Sample Models

```
# Notebook Model with Content
{
    'content': {
        'metadata': {},
        'nbformat': 4,
        'nbformat_minor': 0,
        'cells': [
            {
                'cell_type': 'markdown',
                'metadata': {},
                'source': 'Some **Markdown**',
            },
        ],
    },
    'created': datetime(2015, 7, 25, 19, 50, 19, 19865),
    'format': 'json',
    'last_modified': datetime(2015, 7, 25, 19, 50, 19, 19865),
    'mimetype': None,
    'name': 'a.ipynb',
    'path': 'foo/a.ipynb',
    'type': 'notebook',
    'writable': True,
}

# Notebook Model without Content
{
    'content': None,
    'created': datetime.datetime(2015, 7, 25, 20, 17, 33, 271931),
    'format': None,
```

```
    'last_modified': datetime.datetime(2015, 7, 25, 20, 17, 33, 271931),
    'mimetype': None,
    'name': 'a.ipynb',
    'path': 'foo/a.ipynb',
    'type': 'notebook',
    'writable': True
}
```

API Paths

ContentsManager methods represent the locations of filesystem resources as **API-style paths**. Such paths are interpreted as relative to the root directory of the notebook server. For compatibility across systems, the following guarantees are made:

- Paths are always `unicode`, not `bytes`.

- Paths are not URL-escaped.

- Paths are always forward-slash (/) delimited, even on Windows.

- Leading and trailing slashes are stripped. For example, `/foo/bar/buzz/` becomes `foo/bar/buzz`.

- The empty string (`""`) represents the root directory.

9.1.2 Writing a Custom ContentsManager

The default ContentsManager is designed for users running the notebook as an application on a personal computer. It stores notebooks as .ipynb files on the local filesystem, and it maps files and directories in the Notebook UI to files and directories on disk. It is possible to override how notebooks are stored by implementing your own custom subclass of `ContentsManager`. For example, if you deploy the notebook in a context where you don't trust or don't have access to the filesystem of the notebook server, it's possible to write your own ContentsManager that stores notebooks and files in a database.

Required Methods

A minimal complete implementation of a custom `ContentsManager` must implement the following methods:

`ContentsManager.get`(path[, content, type, ...])	Get a file or directory model.
`ContentsManager.save`(model, path)	Save a file or directory model to path.
`ContentsManager.delete_file`(path)	Delete the file or directory at path.
`ContentsManager.rename_file`(old_path, new_path)	Rename a file or directory.
`ContentsManager.file_exists`([path])	Does a file exist at the given path?
`ContentsManager.dir_exists`(path)	Does a directory exist at the given path?
`ContentsManager.is_hidden`(path)	Is path a hidden directory or file?

9.1.3 Customizing Checkpoints

TODO:

9.1.4 Testing

`notebook.services.contents.tests` includes several test suites written against the abstract Contents API. This means that an excellent way to test a new ContentsManager subclass is to subclass our tests to make them use your ContentsManager.

Note: PGContents is an example of a complete implementation of a custom `ContentsManager`. It stores notebooks and files in PostgreSQL and encodes directories as SQL relations. PGContents also provides an example of how to re-use the notebook's tests.

9.2 File save hooks

You can configure functions that are run whenever a file is saved. There are two hooks available:

- `ContentsManager.pre_save_hook` runs on the API path and model with content. This can be used for things like stripping output that people don't like adding to VCS noise.

- `FileContentsManager.post_save_hook` runs on the filesystem path and model without content. This could be used to commit changes after every save, for instance.

They are both called with keyword arguments:

```
pre_save_hook(model=model, path=path, contents_manager=cm)
post_save_hook(model=model, os_path=os_path, contents_manager=cm)
```

9.2.1 Examples

These can both be added to `jupyter_notebook_config.py`.

A pre-save hook for stripping output:

```python
def scrub_output_pre_save(model, **kwargs):
    """scrub output before saving notebooks"""
    # only run on notebooks
    if model['type'] != 'notebook':
        return
    # only run on nbformat v4
    if model['content']['nbformat'] != 4:
        return

    for cell in model['content']['cells']:
        if cell['cell_type'] != 'code':
            continue
        cell['outputs'] = []
        cell['execution_count'] = None

c.FileContentsManager.pre_save_hook = scrub_output_pre_save
```

A post-save hook to make a script equivalent whenever the notebook is saved (replacing the `--script` option in older versions of the notebook):

```python
import io
import os
from notebook.utils import to_api_path
```

```python
_script_exporter = None


def script_post_save(model, os_path, contents_manager, **kwargs):
    """convert notebooks to Python script after save with nbconvert

    replaces `ipython notebook --script`
    """
    from nbconvert.exporters.script import ScriptExporter

    if model['type'] != 'notebook':
        return

    global _script_exporter
    if _script_exporter is None:
        _script_exporter = ScriptExporter(parent=contents_manager)
    log = contents_manager.log

    base, ext = os.path.splitext(os_path)
    py_fname = base + '.py'
    script, resources = _script_exporter.from_filename(os_path)
    script_fname = base + resources.get('output_extension', '.txt')
    log.info("Saving script /%s", to_api_path(script_fname, contents_manager.root_dir))
    with io.open(script_fname, 'w', encoding='utf-8') as f:
        f.write(script)
c.FileContentsManager.post_save_hook = script_post_save
```

This could be a simple call to `jupyter nbconvert --to script`, but spawning the subprocess every time is quite slow.

9.3 Custom request handlers

The notebook webserver can be interacted with using a well defined RESTful API. You can define custom RESTful API handlers in addition to the ones provided by the notebook. As described below, to define a custom handler you need to first write a notebook server extension. Then, in the extension, you can register the custom handler.

9.3.1 Writing a notebook server extension

The notebook webserver is written in Python, hence your server extension should be written in Python too. Server extensions, like IPython extensions, are Python modules that define a specially named load function, `load_jupyter_server_extension`. This function is called when the extension is loaded.

```python
def load_jupyter_server_extension(nb_server_app):
    """
    Called when the extension is loaded.

    Args:
        nb_server_app (NotebookWebApplication): handle to the Notebook webserver instance.
    """
    pass
```

To get the notebook server to load your custom extension, you'll need to add it to the list of extensions to be loaded. You can do this using the config system. `NotebookApp.server_extensions` is a config variable which is an array of strings, each a Python module to be imported. Because this variable is notebook config, you can set it two different ways, using config files or via the command line.

For example, to get your extension to load via the command line add a double dash before the variable name, and put the Python array in double quotes. If your package is "mypackage" and module is "mymodule", this would look like `jupyter notebook --NotebookApp.server_extensions="['mypackage.mymodule']"`. Basically the string should be Python importable.

Alternatively, you can have your extension loaded regardless of the command line args by setting the variable in the Jupyter config file. The default location of the Jupyter config file is `~/.jupyter/profile_default/jupyter_notebook_config.py`. Then, inside the config file, you can use Python to set the variable. For example, the following config does the same as the previous command line example [1].

```
c = get_config()
c.NotebookApp.server_extensions = [
    'mypackage.mymodule'
]
```

Before continuing, it's a good idea to verify that your extension is being loaded. Use a print statement to print something unique. Launch the notebook server and you should see your statement printed to the console.

9.3.2 Registering custom handlers

Once you've defined a server extension, you can register custom handlers because you have a handle to the Notebook server app instance (`nb_server_app` above). However, you first need to define your custom handler. To declare a custom handler, inherit from `notebook.base.handlers.IPythonHandler`. The example below[1] is a Hello World handler:

```python
from notebook.base.handlers import IPythonHandler


class HelloWorldHandler(IPythonHandler):
    def get(self):
        self.finish('Hello, world!')
```

The Jupyter Notebook server use Tornado as its web framework. For more information on how to implement request handlers, refer to the Tornado documentation on the matter.

After defining the handler, you need to register the handler with the Notebook server. See the following example:

```python
web_app = nb_server_app.web_app
host_pattern = '.*$'
route_pattern = url_path_join(web_app.settings['base_url'], '/hello')
web_app.add_handlers(host_pattern, [(route_pattern, HelloWorldHandler)])
```

Putting this together with the extension code, the example looks like the following:

```python
from notebook.utils import url_path_join
from notebook.base.handlers import IPythonHandler


class HelloWorldHandler(IPythonHandler):
    def get(self):
        self.finish('Hello, world!')


def load_jupyter_server_extension(nb_server_app):
    """
    Called when the extension is loaded.

    Args:
        nb_server_app (NotebookWebApplication): handle to the Notebook webserver instance.
    """
```

```
    web_app = nb_server_app.web_app
    host_pattern = '.*$'
    route_pattern = url_path_join(web_app.settings['base_url'], '/hello')
    web_app.add_handlers(host_pattern, [(route_pattern, HelloWorldHandler)])
```

References: 1. Peter Parente's Mindtrove

9.4 Custom front-end extensions

This describes the basic steps to write a JavaScript extension for the Jupyter notebook front-end. This allows you to customize the behaviour of the various pages like the dashboard, the notebook, or the text editor.

9.4.1 The structure of a front-end extension

Note: The notebook front-end and Javascript API are not stable, and are subject to a lot of changes. Any extension written for the current notebook is almost guaranteed to break in the next release.

A front-end extension is a JavaScript file that defines an AMD module which exposes at least a function called `load_ipython_extension`, which takes no arguments. We will not get into the details of what each of these terms consists of yet, but here is the minimal code needed for a working extension:

```
// file my_extension/main.js

define(function(){

    function load_ipython_extension(){
        console.info('this is my first extension');
    }

    return {
        load_ipython_extension: load_ipython_extension
    };
});
```

Note: Although for historical reasons the function is called `load_ipython_extension`, it does apply to the Jupyter notebook in general, and will work regardless of the kernel in use.

If you are familiar with JavaScript, you can use this template to require any Jupyter module and modify its configuration, or do anything else in client-side Javascript. Your extension will be loaded at the right time during the notebook page initialisation for you to set up a listener for the various events that the page can trigger.

You might want access to the current instances of the various Jupyter notebook components on the page, as opposed to the classes defined in the modules. The current instances are exposed by a module named `base/js/namespace`. If you plan on accessing instances on the page, you should `require` this module rather than accessing the global variable `Jupyter`, which will be removed in future. The following example demonstrates how to access the current notebook instance:

```
// file my_extension/main.js

define([
    'base/js/namespace'
```

```
], function(
    Jupyter
) {
    function load_ipython_extension() {
        console.log(
            'This is the current notebook application instance:',
            Jupyter.notebook
        );
    }

    return {
        load_ipython_extension: load_ipython_extension
    };
});
```

9.4.2 Modifying key bindings

One of the abilities of extensions is to modify key bindings, although once again this is an API which is not guaranteed to be stable. However, custom key bindings are frequently requested, and are helpful to increase accessibility, so in the following we show how to access them.

Here is an example of an extension that will unbind the shortcut `0`, `0` in command mode, which normally restarts the kernel, and bind `0`, `0`, `0` in its place:

```
// file my_extension/main.js

define([
    'base/js/namespace'
], function(
    Jupyter
) {

    function load_ipython_extension() {
        Jupyter.keyboard_manager.command_shortcuts.remove_shortcut('0,0');
        Jupyter.keyboard_manager.command_shortcuts.add_shortcut('0,0,0', 'jupyter-notebook:restart-ke
    }

    return {
        load_ipython_extension: load_ipython_extension
    };
});
```

Note: The standard keybindings might not work correctly on non-US keyboards. Unfortunately, this is a limitation of browser implementations and the status of keyboard event handling on the web in general. We appreciate your feedback if you have issues binding keys, or have any ideas to help improve the situation.

You can see that I have used the **action name** `jupyter-notebook:restart-kernel` to bind the new shortcut. There is no API yet to access the list of all available *actions*, though the following in the JavaScript console of your browser on a notebook page should give you an idea of what is available:

```
Object.keys(require('base/js/namespace').actions._actions);
```

In this example, we changed a keyboard shortcut in **command mode**; you can also customize keyboard shortcuts in **edit mode**. However, most of the keyboard shortcuts in edit mode are handled by CodeMirror, which supports custom key bindings via a completely different API.

9.4.3 Defining and registering your own actions

As part of your front-end extension, you may wish to define actions, which can be attached to toolbar buttons, or called from the command palette. Here is an example of an extension that defines a (not very useful!) action to show an alert, and adds a toolabr button using the full action name:

```javascript
// file my_extension/main.js

define([
    'base/js/namespace'
], function(
    Jupyter
) {
    function load_ipython_extension() {

        var handler = function () {
            alert('this is an alert from my_extension!');
        };

        var action = {
            icon: 'fa-comment-o', // a font-awesome class used on buttons, etc
            help    : 'Show an alert',
            help_index : 'zz',
            handler : handler
        };
        var prefix = 'my_extension';
        var action_name = 'show-alert';

        var full_action_name = Jupyter.actions.register(action, name, prefix); // returns 'my_extens
        Jupyter.toolbar.add_buttons_group([full_action_name]);
    }

    return {
        load_ipython_extension: load_ipython_extension
    };
});
```

Every action needs a name, which, when joined with its prefix to make the full action name, should be unique. Built-in actions, like the `jupyter-notebook:restart-kernel` we bound in the earlier *Modifying key bindings* example, use the prefix `jupyter-notebook`. For actions defined in an extension, it makes sense to use the extension name as the prefix. For the action name, the following guidelines should be considered:

- First pick a noun and a verb for the action. For example, if the action is "restart kernel," the verb is "restart" and the noun is "kernel".

- Omit terms like "selected" and "active" by default, so "delete-cell", rather than "delete-selected-cell". Only provide a scope like "-all-" if it is other than the default "selected" or "active" scope.

- If an action has a secondary action, separate the secondary action with "-and-", so "restart-kernel-and-clear-output".

- Use above/below or previous/next to indicate spatial and sequential relationships.

- Don't ever use before/after as they have a temporal connotation that is confusing when used in a spatial context.

- For dialogs, use a verb that indicates what the dialog will accomplish, such as "confirm-restart-kernel".

9.4.4 Installing and enabling extensions

You can install your nbextension with the command:

```
jupyter nbextension install path/to/my_extension/ [--user|--sys-prefix]
```

The default installation is system-wide. You can use `--user` to do a per-user installation, or `--sys-prefix` to install to Python's prefix (e.g. in a virtual or conda environment). Where my_extension is the directory containing the Javascript files. This will copy it to a Jupyter data directory (the exact location is platform dependent - see Data files).

For development, you can use the `--symlink` flag to symlink your extension rather than copying it, so there's no need to reinstall after changes.

To use your extension, you'll also need to **enable** it, which tells the notebook interface to load it. You can do that with another command:

```
jupyter nbextension enable my_extension/main [--sys-prefix]
```

The argument refers to the Javascript module containing your `load_ipython_extension` function, which is `my_extension/main.js` in this example. There is a corresponding `disable` command to stop using an extension without uninstalling it.

Changed in version 4.2: Added `--sys-prefix` argument

9.5 Customize keymaps

Note: Declarative Custom Keymaps is a provisional feature with unstable API which is not guaranteed to be kept in future versions of the notebook, and can be removed or changed without warnings.

The notebook shortcuts that are defined by jupyter both in edit mode an command mode are configurable in the frontend configuration file `~/.jupyter/nbconfig/notebook.json`. The modification of Keyboard shortcut suffer of several limitations, mainly that your Browser and OS might prevent certain shortcut to work correctly. If this is the case, there are unfortunately not much than can be done. The second issue can arise with keyboard that have a layout different than US English. Again even if we are aware of the issue, there is not much we can do about that.

Shortcut are also limited by the underlying library that handle code and text edition: CodeMirror. If some Keyboard shortcuts are conflicting, the method describe below might not work to create new keyboard shortcuts, especially in the `edit` mode of the notebook.

The 4 sections of interest in `~/.jupyter/nbconfig/notebook.json` are the following:

- `keys.command.unbind`
- `keys.edit.unbind`
- `keys.command.bind`
- `keys.edit.bind`

The first two section describe which default keyboard shortcut not to register at notebook startup time. These are mostly useful if you need to `unbind` a default keyboard shortcut before binding it to a new `command`.

These two first sections apply respectively to the `command` and `edit` mode of the notebook. They take a list of shortcut to `unbind`.

For example, to unbind the shortcut to split a cell at the position of the cursor (`Ctrl-Shift-Minus`)use the following:

```
// file ~/.jupyter/nbconfig/notebook.json

{
  "keys": {
    "edit": {
      "unbind": [
        "Ctrl-Shift-Minus"
      ]
    },
  },
}
```

The last two section describe which new keyboard shortcut to register at notebook startup time, and which actions they trigger.

These two last sections apply respectively to the `command` and `edit` mode of the notebook. They take a dictionary with shortcuts as `keys` and `commands` name as value.

For example, to bind the shortcut `G,G,G` (Press G three time in a row) in command mode, to the command that restart the kernel and run all cells, use the following:

```
// file ~/.jupyter/nbconfig/notebook.json

{
  "keys": {
    "command": {
      "bind": {
        "G,G,G":"jupyter-notebook:restart-kernel-and-run-all-cells"
      }
    }
  },
}
```

The name of the available `commands` can be find by hovering the right end of a row in the command palette.

Contributing to the Jupyter Notebook

If you're reading this section, you're probably interested in contributing to Jupyter. Welcome and thanks for your interest in contributing!

Please take a look at the Contributor documentation, familiarize yourself with using the Jupyter Notebook, and introduce yourself on the mailing list and share what area of the project you are interested in working on.

10.1 General Guidelines

For general documentation about contributing to Jupyter projects, see the Project Jupyter Contributor Documentation.

10.2 Setting Up a Development Environment

For general installation instructions have a look at the Project Jupyter Installation Guide.

10.2.1 Installing Node.js and npm

Building the Notebook from its GitHub source code requires some tools to create and minify JavaScript components and the CSS. Namely, that's Node.js and Node's package manager, `npm`.

If you use `conda`, you can get them with:

```
conda install -c javascript nodejs
```

If you use Homebrew on Mac OS X:

```
brew install node
```

For Debian/Ubuntu systems, you should use the `nodejs-legacy` package instead of the `node` package:

```
sudo apt-get update
sudo apt-get install nodejs-legacy npm
```

You can also use the installer from the Node.js website.

10.2.2 Installing the Jupyter Notebook

Once you have installed the dependencies mentioned above, use the following steps:

```
pip install setuptools pip --upgrade --user
git clone https://github.com/jupyter/notebook
cd notebook
pip install -e . --user
```

If you want the development environment to be available for all users of your system (assuming you have the necessary rights) or if you are installing in a virtual environment, just drop the `--user` option.

10.2.3 Rebuilding JavaScript and CSS

There is a build step for the JavaScript and CSS in the notebook. To make sure that you are working with up-to-date code, you will need to run this command whenever there are changes to JavaScript or LESS sources:

```
python setup.py js css
```

Prototyping Tip

When doing prototyping which needs quick iteration of the Notebook's JavaScript, run this in the root of the repository:

```
npm run build:watch
```

This will cause WebPack to monitor the files you edit and recompile them on the fly.

Git Hooks

If you want to automatically update dependencies, recompile the JavaScript, and recompile the CSS after checking out a new commit, you can install post-checkout and post-merge hooks which will do it for you:

```
git-hooks/install-hooks.sh
```

See `git-hooks/README.md` for more details.

10.3 Running Tests

10.3.1 Python Tests

Install dependencies:

```
pip install -e .[test] --user
```

To run the Python tests, use:

```
nosetests
```

If you want coverage statistics as well, you can run:

```
nosetests --with-coverage --cover-package=notebook notebook
```

10.3.2 JavaScript Tests

To run the JavaScript tests, you will need to have PhantomJS and CasperJS installed:

```
npm install -g casperjs phantomjs@1.9.18
```

Then, to run the JavaScript tests:

```
python -m notebook.jstest [group]
```

where `[group]` is an optional argument that is a path relative to `notebook/tests/`. For example, to run all tests in `notebook/tests/notebook`:

```
python -m notebook.jstest notebook
```

or to run just `notebook/tests/notebook/deletecell.js`:

```
python -m notebook.jstest notebook/deletecell.js
```

10.4 Building the Documentation

To build the documentation you'll need Sphinx and a few other packages.

To install (and activate) a conda environment named `notebook_docs` containing all the necessary packages, use:

```
conda env create -f docs/environment.yml
source activate notebook_docs    # Linux and OS X
activate notebook_docs           # Windows
```

If you want to install the necessary packages with `pip` instead, use (omitting –user if working in a virtual environment):

```
pip install -r docs/doc-requirements.txt --user
```

Once you have installed the required packages, you can build the docs with:

```
cd docs
make html
```

After that, the generated HTML files will be available at `build/html/index.html`. You may view the docs in your browser.

You can automatically check if all hyperlinks are still valid:

```
make linkcheck
```

Windows users can find `make.bat` in the `docs` folder.

You should also have a look at the Project Jupyter Documentation Guide.

Making a Notebook release

This document guides a contributor through creating a release of the Jupyter notebook.

11.1 Check installed tools

Review `CONTRIBUTING.rst`. Make sure all the tools needed to generate the minified JavaScript and CSS files are properly installed.

11.2 Clean the repository

You can remove all non-tracked files with:

```
git clean -xfdi
```

This would ask you for confirmation before removing all untracked files. Make sure the `dist/` folder is clean and avoid stale build from previous attempts.

11.3 Create the release

1. Update version number in `notebook/_version.py`.

2. Run this command:

   ```
   python setup.py jsversion
   ```

   It will modify (at least) `notebook/static/base/js/namespace.js` which makes the notebook version available from within JavaScript.

3. Commit and tag the release with the current version number:

   ```
   git commit -am "release $VERSION"
   git tag $VERSION
   ```

4. You are now ready to build the `sdist` and `wheel`:

   ```
   python setup.py sdist --formats=zip,gztar
   python setup.py bdist_wheel
   ```

5. You can now test the `wheel` and the `sdist` locally before uploading to PyPI. Make sure to use twine to upload the archives over SSL.

```
twine upload dist/*
```

6. If all went well, change the `notebook/_version.py` back adding the `.dev` suffix.

7. Push directly on master, not forgetting to push `--tags` too.

Developer FAQ

1. How do I install a prerelease version such as a beta or release candidate?

```
python -m pip install notebook --pre --upgrade
```

View the original notebooks on nbviewer

Examples

The following notebooks have been rendered for your convenience.

13.1 What is the Jupyter Notebook?

13.1.1 Introduction

The Jupyter Notebook is an **interactive computing environment** that enables users to author notebook documents that include: - Live code - Interactive widgets - Plots - Narrative text - Equations - Images - Video

These documents provide a **complete and self-contained record of a computation** that can be converted to various formats and shared with others using email, Dropbox, version control systems (like git/GitHub) or nbviewer.jupyter.org.

Components

The Jupyter Notebook combines three components:

- **The notebook web application**: An interactive web application for writing and running code interactively and authoring notebook documents.

- **Kernels**: Separate processes started by the notebook web application that runs users' code in a given language and returns output back to the notebook web application. The kernel also handles things like computations for interactive widgets, tab completion and introspection.

- **Notebook documents**: Self-contained documents that contain a representation of all content visible in the notebook web application, including inputs and outputs of the computations, narrative text, equations, images, and rich media representations of objects. Each notebook document has its own kernel.

13.1.2 Notebook web application

The notebook web application enables users to:

- **Edit code in the browser**, with automatic syntax highlighting, indentation, and tab completion/introspection.

- **Run code from the browser**, with the results of computations attached to the code which generated them.

- See the results of computations with **rich media representations**, such as HTML, LaTeX, PNG, SVG, PDF, etc.

- Create and use **interactive JavaScript widgets**, which bind interactive user interface controls and visualizations to reactive kernel side computations.

- Author **narrative text** using the Markdown markup language.

- Build **hierarchical documents** that are organized into sections with different levels of headings.

- Include mathematical equations using **LaTeX syntax in Markdown**, which are rendered in-browser by Math-Jax.

13.1.3 Kernels

Through Jupyter's kernel and messaging architecture, the Notebook allows code to be run in a range of different programming languages. For each notebook document that a user opens, the web application starts a kernel that runs the code for that notebook. Each kernel is capable of running code in a single programming language and there are kernels available in the following languages:

- Python(https://github.com/ipython/ipython)

- Julia (https://github.com/JuliaLang/IJulia.jl)

- R (https://github.com/takluyver/IRkernel)

- Ruby (https://github.com/minrk/iruby)

- Haskell (https://github.com/gibiansky/IHaskell)

- Scala (https://github.com/Bridgewater/scala-notebook)

- node.js (https://gist.github.com/Carreau/4279371)

- Go (https://github.com/takluyver/igo)

The default kernel runs Python code. The notebook provides a simple way for users to pick which of these kernels is used for a given notebook.

Each of these kernels communicate with the notebook web application and web browser using a JSON over Ze-roMQ/WebSockets message protocol that is described here. Most users don't need to know about these details, but it helps to understand that "kernels run code."

13.1.4 Notebook documents

Notebook documents contain the **inputs and outputs** of an interactive session as well as **narrative text** that accompanies the code but is not meant for execution. **Rich output** generated by running code, including HTML, images, video, and plots, is embeddeed in the notebook, which makes it a complete and self-contained record of a computation.

When you run the notebook web application on your computer, notebook documents are just **files on your local filesystem with a ''.ipynb'' extension**. This allows you to use familiar workflows for organizing your notebooks into folders and sharing them with others.

Notebooks consist of a **linear sequence of cells**. There are four basic cell types:

- **Code cells:** Input and output of live code that is run in the kernel

- **Markdown cells:** Narrative text with embedded LaTeX equations

- **Heading cells:** 6 levels of hierarchical organization and formatting

- **Raw cells:** Unformatted text that is included, without modification, when notebooks are converted to different formats using nbconvert

Internally, notebook documents are **'JSON <http://en.wikipedia.org/wiki/JSON>'__ data** with **binary values 'base64 <http://en.wikipedia.org/wiki/Base64>'__** encoded. This allows them to be **read and manipulated programmatically** by any programming language. Because JSON is a text format, notebook documents are version control friendly.

Notebooks can be exported to different static formats including HTML, reStructeredText, LaTeX, PDF, and slide shows (reveal.js) using Jupyter's `nbconvert` utility.

Furthermore, any notebook document available from a **public URL on or GitHub can be shared** via nbviewer. This service loads the notebook document from the URL and renders it as a static web page. The resulting web page may thus be shared with others **without their needing to install the Jupyter Notebook**.

13.2 Notebook Basics

13.2.1 The Notebook dashboard

When you first start the notebook server, your browser will open to the notebook dashboard. The dashboard serves as a home page for the notebook. Its main purpose is to display the notebooks and files in the current directory. For example, here is a screenshot of the dashboard page for the `examples` directory in the Jupyter repository:

The top of the notebook list displays clickable breadcrumbs of the current directory. By clicking on these breadcrumbs or on sub-directories in the notebook list, you can navigate your file system.

To create a new notebook, click on the "New" button at the top of the list and select a kernel from the dropdown (as seen below). Which kernels are listed depend on what's installed on the server. Some of the kernels in the screenshot below may not exist as an option to you.

Notebooks and files can be uploaded to the current directory by dragging a notebook file onto the notebook list or by the "click here" text above the list.

The notebook list shows green "Running" text and a green notebook icon next to running notebooks (as seen below). Notebooks remain running until you explicitly shut them down; closing the notebook's page is not sufficient.

To shutdown, delete, duplicate, or rename a notebook check the checkbox next to it and an array of controls will appear at the top of the notebook list (as seen below). You can also use the same operations on directories and files when applicable.

To see all of your running notebooks along with their directories, click on the "Running" tab:

This view provides a convenient way to track notebooks that you start as you navigate the file system in a long running notebook server.

13.2.2 Overview of the Notebook UI

If you create a new notebook or open an existing one, you will be taken to the notebook user interface (UI). This UI allows you to run code and author notebook documents interactively. The notebook UI has the following main areas:

- Menu
- Toolbar
- Notebook area and cells

The notebook has an interactive tour of these elements that can be started in the "Help:User Interface Tour" menu item.

13.2.3 Modal editor

Starting with IPython 2.0, the Jupyter Notebook has a modal user interface. This means that the keyboard does different things depending on which mode the Notebook is in. There are two modes: edit mode and command mode.

Edit mode

Edit mode is indicated by a green cell border and a prompt showing in the editor area:

When a cell is in edit mode, you can type into the cell, like a normal text editor.

Enter edit mode by pressing `Enter` or using the mouse to click on a cell's editor area.

Command mode

Command mode is indicated by a grey cell border with a blue left margin:

When you are in command mode, you are able to edit the notebook as a whole, but not type into individual cells. Most importantly, in command mode, the keyboard is mapped to a set of shortcuts that let you perform notebook and cell actions efficiently. For example, if you are in command mode and you press `c`, you will copy the current cell - no modifier is needed.

Don't try to type into a cell in command mode; unexpected things will happen!

Enter command mode by pressing `Esc` or using the mouse to click *outside* a cell's editor area.

13.2.4 Mouse navigation

All navigation and actions in the Notebook are available using the mouse through the menubar and toolbar, which are both above the main Notebook area:

The first idea of mouse based navigation is that **cells can be selected by clicking on them.** The currently selected cell gets a grey or green border depending on whether the notebook is in edit or command mode. If you click inside a cell's editor area, you will enter edit mode. If you click on the prompt or output area of a cell you will enter command mode.

If you are running this notebook in a live session (not on http://nbviewer.jupyter.org) try selecting different cells and going between edit and command mode. Try typing into a cell.

The second idea of mouse based navigation is that **cell actions usually apply to the currently selected cell**. Thus if you want to run the code in a cell, you would select it and click the

button in the toolbar or the "Cell:Run" menu item. Similarly, to copy a cell you would select it and click the

button in the toolbar or the "Edit:Copy" menu item. With this simple pattern, you should be able to do most everything you need with the mouse.

Markdown and heading cells have one other state that can be modified with the mouse. These cells can either be rendered or unrendered. When they are rendered, you will see a nice formatted representation of the cell's contents. When they are unrendered, you will see the raw text source of the cell. To render the selected cell with the mouse, click the

button in the toolbar or the "Cell:Run" menu item. To unrender the selected cell, double click on the cell.

13.2.5 Keyboard Navigation

The modal user interface of the Jupyter Notebook has been optimized for efficient keyboard usage. This is made possible by having two different sets of keyboard shortcuts: one set that is active in edit mode and another in command mode.

The most important keyboard shortcuts are `Enter`, which enters edit mode, and `Esc`, which enters command mode.

In edit mode, most of the keyboard is dedicated to typing into the cell's editor. Thus, in edit mode there are relatively few shortcuts. In command mode, the entire keyboard is available for shortcuts, so there are many more. The `Help->``Keyboard Shortcuts``` dialog lists the available shortcuts.

We recommend learning the command mode shortcuts in the following rough order:

1. Basic navigation: `enter`, `shift-enter`, `up/k`, `down/j`
2. Saving the notebook: `s`
3. Change Cell types: `y`, `m`, `1-6`, `t`
4. Cell creation: `a`, `b`
5. Cell editing: `x`, `c`, `v`, `d`, `z`
6. Kernel operations: `i`, `0` (press twice)

13.3 Running Code

First and foremost, the Jupyter Notebook is an interactive environment for writing and running code. The notebook is capable of running code in a wide range of languages. However, each notebook is associated with a single kernel. This notebook is associated with the IPython kernel, therefor runs Python code.

13.3.1 Code cells allow you to enter and run code

Run a code cell using `Shift-Enter` or pressing the

button in the toolbar above:

```
In [2]: a = 10
```

```
In [3]: print(a)
```

```
10
```

There are two other keyboard shortcuts for running code:

- `Alt-Enter` runs the current cell and inserts a new one below.
- `Ctrl-Enter` run the current cell and enters command mode.

13.3.2 Managing the Kernel

Code is run in a separate process called the Kernel. The Kernel can be interrupted or restarted. Try running the following cell and then hit the

button in the toolbar above.

```
In [4]: import time
        time.sleep(10)
```

If the Kernel dies you will be prompted to restart it. Here we call the low-level system libc.time routine with the wrong argument via ctypes to segfault the Python interpreter:

```
In [5]: import sys
        from ctypes import CDLL
        # This will crash a Linux or Mac system
        # equivalent calls can be made on Windows

        # Uncomment these lines if you would like to see the segfault

        # dll = 'dylib' if sys.platform == 'darwin' else 'so.6'
        # libc = CDLL("libc.%s" % dll)
        # libc.time(-1)  # BOOM!!
```

13.3.3 Cell menu

The "Cell" menu has a number of menu items for running code in different ways. These includes:

- Run and Select Below
- Run and Insert Below
- Run All
- Run All Above
- Run All Below

13.3.4 Restarting the kernels

The kernel maintains the state of a notebook's computations. You can reset this state by restarting the kernel. This is done by clicking on the

in the toolbar above.

13.3.5 sys.stdout and sys.stderr

The stdout and stderr streams are displayed as text in the output area.

```
In [6]: print("hi, stdout")
```

```
hi, stdout
```

```
In [7]: from __future__ import print_function
        print('hi, stderr', file=sys.stderr)
```

```
hi, stderr
```

13.3.6 Output is asynchronous

All output is displayed asynchronously as it is generated in the Kernel. If you execute the next cell, you will see the output one piece at a time, not all at the end.

```
In [8]: import time, sys
        for i in range(8):
            print(i)
            time.sleep(0.5)
```

```
0
1
2
3
4
5
6
7
```

13.3.7 Large outputs

To better handle large outputs, the output area can be collapsed. Run the following cell and then single- or double-click on the active area to the left of the output:

```
In [9]: for i in range(50):
            print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
```

```
37
38
39
40
41
42
43
44
45
46
47
48
49
```

Beyond a certain point, output will scroll automatically:

```
In [10]: for i in range(500):
             print(2**i - 1)

0
1
3
7
15
31
63
127
255
511
1023
2047
4095
8191
16383
32767
65535
131071
262143
524287
1048575
2097151
4194303
8388607
16777215
33554431
67108863
134217727
268435455
536870911
1073741823
2147483647
4294967295
8589934591
17179869183
34359738367
```

```
68719476735
137438953471
274877906943
549755813887
1099511627775
2199023255551
4398046511103
8796093022207
17592186044415
35184372088831
70368744177663
140737488355327
281474976710655
562949953421311
1125899906842623
2251799813685247
4503599627370495
9007199254740991
18014398509481983
36028797018963967
72057594037927935
144115188075855871
288230376151711743
576460752303423487
1152921504606846975
2305843009213693951
4611686018427387903
9223372036854775807
18446744073709551615
36893488147419103231
73786976294838206463
147573952589676412927
295147905179352825855
590295810358705651711
1180591620717411303423
2361183241434822606847
4722366482869645213695
9444732965739290427391
18889465931478580854783
37778931862957161709567
75557863725914323419135
151115727451828646838271
302231454903657293676543
604462909807314587353087
1208925819614629174706175
2417851639229258349412351
4835703278458516698824703
9671406556917033397649407
19342813113834066795298815
38685626227668133590597631
77371252455336267181195263
154742504910672534362390527
309485009821345068724781055
618970019642690137449562111
```

```
12379400392853802748991242 23
24758800785707605497982484 47
49517601571415210995964968 95
99035203142830421991929937 91
19807040628566084398385987 583
39614081257132168796771975 167
79228162514264337593543950 335
15845632502852867518708790 0671
31691265005705735037417580 1343
63382530011411470074835160 2687
12676506002282294014967032 05375
25353012004564588029934064 10751
50706024009129176059868128 21503
10141204801825835211973625 643007
20282409603651670423947251 286015
40564819207303340847894502 572031
81129638414606681695789005 144063
16225927682921336339157801 0288127
32451855365842672678315602 0576255
64903710731685345356631204 1152511
12980742146337069071326240 82305023
25961484292674138142652481 64610047
51922968585348276285304963 29220095
10384593717069655257060992 658440191
20769187434139310514121985 316880383
41538374868278621028243970 633760767
83076749736557242056487941 267521535
16615349947311448411297588 2535043071
33230699894622896822595176 5070086143
66461399789245793645190353 0140172287
13292279957849158729038070 60280344575
26584559915698317458076141 20560689151
53169119831396634916152282 41121378303
10633823966279326983230456 482242756607
21267647932558653966460912 964485513215
42535295865117307932921825 928971026431
85070591730234615865843651 857942052863
17014118346046923173168730 3715884105727
34028236692093846346337460 7431768211455
68056473384187692692674921 4863536422911
13611294676837538538534984 29727072845823
27222589353675077077069968 59454145691647
54445178707350154154139937 18908291383295
10889035741470030830827987 437816582766591
21778071482940061661655974 8756331165533183
43556142965880123323311949 7512663310663 67
87112285931760246646623899 502532662132735
17422457186352049329324779 9005065324265471
34844914372704098658649559 8010130648530943
69689828745408197317299119 60202612970618 87
13937965749081639463459823 920405225941237 75
27875931498163278926919647 84081045188247551
55751862996326557853839295 68162090376495103
11150372599265311570767859 1363241807529902 07
```

```
223007451985306231415357182726483615059804415
446014903970612462830714365452967230119608311
892029807941224925661428730905934460239216633
178405961588244985132285746181186892047843327
356811923176489970264571492362373784095686655
713623846352979940529142984724747568191373311
142724769270595988105828596944949513638274623
285449538541191976211657193889899027276549324747
570899077082383952423314387779798054553098649495
114179815416476790484662877555959610910619729911
228359630832953580969325755111919221821239459833
456719261665907161938651510223838443642478919677
913438523331814323877303020447676887284957839355
182687704666362864775460604089535377456991567871
365375409332725729550921208179070754913983135743
730750818665451459101842416358141509827966271487
146150163733090291820368483271628301965593254297555
292300327466180583640736966543256603931186508951
584600654932361167281473933086513207862373017190333
116920130986472233456294786617302641572474603438077
233840261972944466912589573234605283144949206876155
467680523945888933825179146469210566289898413752311
935361047891777867650358292938421132579796827504633
187072209578355573530071658587684226515959365500927
374144419156711147060143317175368453031918731001855
748288838313422294120286634350736906006383746200371111
149657767662684458824057326870147381212767492400742333
2993155353253689176481146537402947624255349848014847
598631070650737835296229307480589524851069969602969555
119726214130147567059245861496117904970213993920593911
23945242826029513411849172299223580994042798784118783
478904856520590268236983445984471619880855975682375677
957809713041180536473966891968943239761711951364751355
191561942608236107294793378393788647952342390272950271
38312388521647221458958675678757729590468478054590054343
766247770432944429179173513575154591809369561091801087
153249554086588885835834702715030918361873912218360217555
30649910817317777167166940543006183672374782443672043511
61299821634635554334333881086012367344749564887344087033
122599643269271108668667762172024734689499129774688174077
245199286538542217337335524344049469378998259549376348155
490398573077084434674671048688098938757996519098752696311
980797146154168869349342097376197877515993038197505392633
196159429230833773869868419475239575503198607639501078527
392318858461667547739736838950479151006397215279002157055
784637716923335095479473677900958302012794430558004314111
156927543384667019095894735580191660402558861116008628223
313855086769334038191789471160383320805117772232017256447
627710173538668076383578942320766641610235544446403451289555
125542034707733615276715788464153328322047108892806902579111
251084069415467230553431576928306656644094217778561380515833
502168138830934461106863153856613313288188435557122761031677
10043362776186892221372630771322662657637687111424552206335
200867255532373784442745261542645325315275374222849104412671
```

```
40173451106474756888549052308529065063055074844569820882534380346902212949513777098104617058130126110149689139641765068716069380442589902755419620923411626025222029937827928353013753213876088517980551083924184682325205044405987565585670602751642775217703596110216784836936465041008881197513117134120550312855504354071922204335696738729300820177623950262342682411007257110087081438444086713934774586016403552479005246853648220155142201741628768881734278695491720328071049580104937072964403110284403483257537763468557390983440656142099160209874145928806320568806966515075526937114781966881312284198320419748291857612741137613933030151053874229563933762624568396640839496583715225582275227866060302107748459127867525249136793281678993167430451116455045573212060421549691825573505049827358656335798633486090233291009114642412084309938365114701009965471731267159726697218047658201822928482416861987673022940201993094346253431945339443609513164036458569648337239753460458804039861886925068638906788872191263280729171392966744795069209176080797237738501372778135777443835265614583427859334895901384183521615944754770027455627155488767105312291668557186697918027683670432318895095400549111254310977535210624583337114373395836055367340864637790190801098222508621955071421249166674228746791672110734681729275580381602196445017243910143842498333348457493583344221469363458551160763204392890034487820287168499666669691498716668844293872691710232152640878578006897564057533369993333393829974333376885877453834204643052817571560137951281151673998666678765994866675377175490766840928610563514312027590256230313479973333575319897333507543509815336818572112702862405518051246072695994666715063979466701508701963067363714442254057248110361024921553919893334301279589334030174039261347274288845081144962207220498431107839786668602559178668060348078522694548577690162289924414440996863215679573337205118357336120696157045389097155380324579848828881993727431359146674410236714672241392314090778194310760649159697657763987455862718293348820473429344482784628181556388621521298319395315527974911172543658669764094685868896556925636311277724304259663879063105594982334508731733952818937173779311385127262255544860851932775812621118996477690174634679056378743475586227702545245110897217038655516252422379929513803492693581127574869511724554050904902217944340773110325048447598591276069853871622551497390234491081018098044358886815462206500968951971835521397077432451029947804689821620361960887177736309244130019379039436711042794154864902059895609379643240723921774355472618488260038758078873522085588309729804119791218759286481447843548710945236976520077516157747144171176619459608239582437518572962895687097421890473953040155032315494388342353238919216479164875037145925791374194843780947906080310064630988717668470647783843295832975007429185158274838968756189581216062012926197753533694129556768659166595001485837031654967793751237916243212402585239551706738825911353731833190002971674063309935587502475832486424805170479103141347765182270746366663800059433481266198711750049516649728496103409582072826955303645414927333276001188669625323974235000990332994569922068191641556539106072908298546665520023773392506479484700019806659891398441363832831113078212145816597093331040047546785012958969400039613319782796882727665663226156424291633194186620800950935700259179388000792266395655937654553313274523128485832663883733241601901871400518358776001584532791311875309106626559046256971665327767466483203803742801036717552003169065582623750618213253111809251394333065553493296640760748560207343510400633813116524750123642650623361850278866613110698659328152149712041468702080126762623304950024728530124
```

```
72370055773322622139731865630429942408293740416025352524660990004945706024951
44740111546645244279463731260859884816587480832050705049321980009891412049912
89480223093290488558927462521719769633174961664101410098643960019782824099835
78960446186580977117854925043439539266349923328202820197287920039565648199671
15792089237316195423570985008687907853269984665640564039457584007913129639935
23158417847463239084714197001737581570653996933128112807891516801582625927987146
31683569492647816942839400347516314130799386625622561578303360316525185597439
26336713898529563388567880069503262826159877325124512315660672063305037119487
18526734277970591267771357601390065256523197546502490246313213441266100742389753
70534685559411825355427152027801305130463950930049804926264268825322014847795170
41069371118823650710854304055602610260927901860099609852528537650644029695590314
82138742237647301421708608111205220521855803720199219705057075301288059391180729
64277484475294602843417216224104410437116074403984394101141506025761187823615592
85549689505892056868344324448208820874232148807968788202283012051522375647231118
57109937901178411373668864889641764174846429761593757640456602410304475129446323
71421987580235682274733772977928352834969285952318751528091320482060895025889274
74284397516047136454946754595585670566993857190463750305618264096412179005177855
94856879503209427290989350919117134113398771438092750061123652819282435801035571189
71375900641885458197870183823426822679754287618550012224730563856487160207114233
79427518012837709163957403676468536453595085752371000244494611277129743204142284775
88550360256754183279148073529370729071901715047420004889892225542594864082845695151
77100720513508366558296147058741458143803430094840009779784451085189728165691391303
54201441027016733116592294117482916287606860189680019559568902170379456331382783607
08402882054033466233184588234965832575213720379360039119137804340758912662765567121
41680576410806693246636917646993166515042744075872007823827560868151782532553113524
28336115282161338649327383529398633303008548815174401564765512173630356506510622714
85667223056432267729865476705879726660601709763034880312953102434726071301302124543971
33444611286453545973095341175945332120341952606976062590620486945214260260424908719
42668892225729070919461906823518906642406839052139521251812409738904285205208498175388
53377844514581418389238136470378132848136781042790425036248194778057041041699635177706
75568902916283677847627294075626569627356208558085007249638955617140820833992703155413
51137805832567355695254588151253139254712417116170014499277911234281641667985407310827
02275611665134711390509176302506278509424834232340028998555822468563283335970815621654
04551223330269422781018352605012557018849668464680057997111644937126566671941631124330
80910244666053884556203670521002511403769933692936011599422328987425313334388326324866
16182048933210776911240734104200502280753986738587202319884465797485062666877665274973
23236409786642155382248146820840100456150797347717440463976893159497012533375533055994
64647281957328431076449629364168020091230159469543488092795378631899402506675106611119
89292945639146568621528992587283360401824603189390869761855907572637988050133502132223397
85858912782931372430579851745667208036492063787817395237118151452757976100267004264447795
71717825565862744861159703491334416072984127575634790474236302905519522005340085288951591
43435651131725489722319406982668832145968255151269580948472605811039044010680170577913182
86871302263450979444638813965337664291936510302539161896945211622078088021360341155836365
73742604526901958889277627930675328583873020605078323793890423244156176042720682311671273
14748520905380391777855525586135065716774604121015664758778084648831235208544136462335254
62949704181076078355571105117227013143354920824203132951755616929766247041708827292467150
92589940836215215671114221023445402628670984164840626590351123385953249408341765458493431
01851798816724304313422284420468908052573419683296812531807022467719064988166835309169868
20370359763344860862684456884093781610514683936659362506361404493543812997633367061833973
40740719526689721725368913768187563221029367873318725012722808987087625995266734123667947
81481439053379443450737827536375126442058735746637450025445617974175251990533468247335895
16296287810675888690147565507275025288411747149327490005089123594835050398106693649469471
32592575621351777380295131014550050576823494298654980010178247189670100796213387298934358
65185151242703554760590262029100101153646988597309960020356494379340201592426774597868716
```

```
130370302485407109521180524058200202307293977194619920040712988758680403184853549195737432
260740604970814219042361048116400404614587954389239840081425977517360806369707098391474864
521481209941628438084722096232800809229175908778479680162851955034721612739414196782949728
104296241988325687616944419246560161845835181755695936032570391006944322547882839356589945
205892483976651375233888838493120323691670363511391872065140782013888645095765678713179891
417184967953302750467777676986240647383340727022783744130281564027777290191531357426359782
834369935906605500935555353972481294766814540455674882605631280555458038306271485271956
166873987181321100187111070794496258953336290809113497652112625611110916076612542970543913
333747974362642200374222141588992517906672581618226995304225251222221832153225085941087826
667495948725284400748444283177985035813345163236453990608450502444443664306450171882175652
133499189745056880149688856635597007162669032647290798121690100488887328612900343764351304
266998379490113760299377713271194014325338065294581596243380200977774657225800687528702608
533996758980227520598755426542388028650676130589163192486760401955554931445160137505740521
106799351796045504119751085308477605730135226117832638497352080391110986289032027501148104
213598703592091008239502170616955211460270452235665276994704160782221972578064055002296208
427197407184182016479000434123391042292054090447133055398940832156444394515612811000459241
854394814368364032958008682467820845841081808942661107978816643128887890312256220009184834
170878962873672806591601736493564169168216361788532221595763328625777578062451244001836966
341757925747345613183203472987128338336432723577064443191526657251555156124902488003673933
683515851494691226366406945974256676672865447154128886383053314503110312249804976007347867
136703170298938245273281389194851335334573089430825777276610662900622062449960995201469573
273406340597876490546562778389702670669146178861651554553221325801244124899921990402939147
546812681195752981093125556779405341338292357723303109106442651602488249799843980805878294
109362536239150596218625111355881068267658471544660621821288530320497649959968796161175658
218725072478301192437250222711762136535316943089321243642577060640995299919937592322351317
437450144956602384874500445423524273070633886178642487285154121281990599839875184644702635
874900289913204769749000890847048546141267772357284974570308242563981199679750369289405270
174980057982640953949800178169409709228253554471456994914061648512796239935950073857881054
349960115965281907899600356338819418456507108942913989828123297025592479871900147715762108
699920231930563815799200712677638836913014217885827979656246594051184959743800295431524216
139984046386112763159840142535527767382602843577165595931249318810236991948760059086304843
279968092772225526319680285071055534765205687154331191862498637620473983897520118172609686
559936185544451052639360570142111069530411374308662383724997275240947967795040236345219373
111987237108890210527872114028422213906082274861732476744999455048189593559008047269043874
223974474217780421055744228056844427812164549723464953489998910096379187118016094538087749
447948948435560842111488456113688855624329099446929906979997820192758374236032189076175498
895897896871121684222976912227377711248658198893859813959995640385516748472064378152350997
179179579374224336844595382445475542249731639778771962791999128077103349694412875630470199
358359158748448673689190764890951084499463279557543925583998256154206699388825751260940398
716718317496897347378381529781902168998926559115087851167996512308413398777651502521880797
143343663499379469475676305956380433799785311823017570233599302461682679755530300504376159
286687326998758938951352611912760867599570623646035140467198604923365359511060601008752319
573374653997517877902705223825521735199141247292070280934397209846730719022121202017504638
114674930799503575580541044765104347039828249458414056186879441969346143804424240403500927
229349861599007151161082089530208694079656498916828112373758883938692287608848480807001855
458699723198014302322164179060417388159312997833656224747517767877384575217696961614003710
917399446396028604644328358120834776318625995667312449495035535754769150435393923228007421
183479889279205720928865671624166955263725199133462489899007107150953830087078784645601484
366959778558411441857731343248333910527450398266924979798014214301907660174157569291202968
733919557116822883715462686496667821054900796533849959596028428603815320348315138582405936
146783911423364576743092537299333564210980159306769991919205685720763064069663027716481187
293567822846729153486185074598667128421960318613539983838411371441526128139326055432962374
587135645693458306972370149197334256843920637227079967676822742883052256278652110865924749
117427129138691661394474029839466851368784127445415993535364548576610451255730422173184949
```

```
2348542582773833227889480596789337027375682548908319870707290971532209025114608443463698998
4697085165547666455778961193578674054751365097816639741414581943064418050229216886927397990
9394170331095332911557922387157348109502730195633279482829163886128836100458433773854795993
1878834066219066582311584477431469621900546039126655896565832777225767220091686754770959198
3757668132438133164623168954862939243801092078253311793131665554451534440183373509541918397
7515336264876266329246337909725878487602184156506623586263331108903068880366747019083836794
1503067252975253265849267581945175697520436831301324717252666221780613776073349403816767358
3006134505950506531698535163890351395040873662602649434505332443561227552146698807633534717
6012269011901013063397070327780702790081747325205298869010664887122455104293397615267069435
1202453802380202612679414065556140558016349465041059773802132977244910205867952305341388
2404907604760405225358828131112281116032698930082119547604265954848982041717359046106827774
4809815209520810450717656262224562232065397860164239095208531909697964083434718092213655548
9619630419041620901435312524449124464130795720328478190417063819395928166869436184427311097
1923926083808324180287062504889824892826159144065695638083412763879185633373887236885462219
3847852167616648360574125009779649785652318288131391276166825527758371266747774473770924438
7695704335233296721148250019559299571304636576262782552333651055516742533495548947541848877
1539140867046659344229650003911859914260927315252555651046673021110334850669910978950836977
3078281734093318688459300007823719828521854630505113020933460422206697013398219579016739551
6156563468186637376918600015647439657043709261010226041866920844413394026796439158033479102
1231312693637327475383720003129487931408741852202045208373384168882678805359287831606695820
2462625387274654950767440006258975862817483704404090416746768337765357610718575663213391640
4925250774549309901534880012517951725634967408808180833493536675530715221437151326426783281
9850501549098619803069760025035903451269934817616361666987073351061430442874302652853566563
1970100309819723960613952005007180690253986963523272333397414670212286088574860530570713312
3940200619639447921227904010014361380507973927046544666794829340424572177149721061141426625
7880401239278895842455808020002872276101594785409308933358965868084914435429944212228285325
1576080247855779168491161604005744552203189570818617866717931736169828870859888424456570650
3152160495711558336982323208011489104406379141637235733435863472339657741719776848913141300
6304320991423116673964646416022978208812758283274471466871726944679315483439553697826282600
1260864198284623334792929283204595641762551656654894293374345388935863096687910739565256520
2521728396569246669585858566409191283525103313309788586748690777871726193375821479130513040
5043456793138493339171717132818382567050206626619577173497381555743452386751642958261026080
1008691358627698667834343426563676513410041325323915434699476311148690477350328591652205210
2017382717255397335668686853127353026820082650647830869398952622297380954700657183304410432
4034765434510794671337373706254706053640165301295661738797905244594761909401314366608820864
8069530869021589342674747412509412107280330602591323477595810489189523818802628733217641729
1613906173804317868534949482501882421456066120518264695519162097837904763760525746643528345
3227812347608635737069889896500376484291213224103652939103832419567580952752105149328705669
6455624695217271474139797930007529685824264482073058782076648391351619055042102986574113383
1291124939043454294827959586001505937164852896414611756415329678270323811008420597314822670
2582249878086908589655919172003011874329705792829223512830659356540647622016841194629645353
5164499756173817179311838344006023748659411585658447025661318713081295244033682389259290700
1032889995123476343586236766880120474973188231713168940513226374261625904880673647785185814
2065779990246952687172473533760240949946376463426337881026452748523251809761347295570371628
4131559980493905374344947067520481899892752926852675762052905497046503619522694591140743256
8263199609878107486898941350409637997855058537053515241058109940930072390453891822814865130
1652639921975621497379788270081927599571011707410703048211621988186014478090778364562973020
3305279843951242994759576540163855199142023414821406096423243976372028956181556729125946052
6610559687902485989519153080327710398284046829642812192846487952744057912363113458251892104
1322111937580497197903830616065542079656809365928562438569297590548811582472622691650378420
2644223875160994395807661232131084159313618731857124877138595181097623164945245383300756841
5288447750321988791615322464262168318627237463714249754277190362195246329890490766601513681
1057689550064397758323064492852433663725447492742849950855438072439049265978098153320302736
2115379100128795516646128985704867327450894985485699901710876144878098531956196306640605473
```

```
4230758200257591033292225797140973465490178997097139980342175228975619706391239261328121094
8461516400515182066584515942819469309803579941942799606843504579512394127824785226562421893
1692303280103036413316903188563893861960715988388559921368700915902478825564957045312484378
3384606560206072826633806377127787723921431976777119842737401831804957651129914090624968757
6769213120412145653267612754255575447842863953554239685474803663609915302259828181249937514
1353842624082429130653522550851115089568572790710847937094960732721983060451965636249987502
2707685248164858261307045101702230179137145581421695874189921465443966120903931272499975005
5415370496329716522614090203404460358274291162843391748379842930887932241807862544999950011
1083074099265943304522818040680892071654858232568678349675968586177586448361572508999990002
2166148198531886609045636081361784143309716465137356699351937172355172896723145017999998000
4332296397063773218091272162723568286619432930274713398703874344710345793446290035999960009
8664592794127546436182544325447136573238865860549426797407748689420691586892580071999920019
1732918558825509287236508865089427314647773172109885359481549737884138317378516014399984001
3465837117651018574473017730178854629295546344219770718963099475768276634757032028799968001
6931674235302037148946035460357709258591092688439541437926198951536553269514064057599936015
1386334847060407429789207092071541851718218537687908287585239790307310653902812811519987201
2772669694120814859578414184143083703436437075375816575170479580614621307805625623039974406
5545339388241629719156828368286167406872874150751633150340959161229242615611251246079948812
1109067877648325943831365673657233481374574830150326630068191832245848523122250249215989762
2218135755296651887662731347314466962749149660300653260136383664491697046244500498431979524
4436271510593303775325462694628933925498299320601306520272767328983394092489000996863959049
8872543021186607550650925389257867850996598641202613040545534657966788184978001993727918099
1774508604237321510130185077851573570199319728240522608109106931593357636995600398745583619
3549017208474643020260370155703147140398639456481045216218213863186715273991200797491167239
7098034416949286040520740311406294280797278912962090432436427726373430547982401594982334479
1419606883389857208104148062281258856159455782592418086487285545274686109596480318996466895
2839213766779714416208296124562517712318911565184836172974571090549372219192960637992933791
5678427533559428832416592249125035424637823130369672345949142181098744438385921275985867583
1135685506711885766483318449825007084927564626073934469189828436219748887677184255197173510
2271371013423771532966636899650014169855129252147868938379656872439497775354368510394347033
4542742026847543065933273799300028339710258504295737876759313744878995550708737020788694060
9085484053695086131866547598600056679420517008591475753518627489757991101417474041577388133
1817096810739017226373309519720011335884103401718295150703725497951598220283494808315477620
3634193621478034452746619039440022671768206803436590301407450995903196440566989616630955253
7268387242956068905493238078880045343536413606873180602814901991806392881133979233261910507
1453677448591213781098647615776009068707282721374636120562980398361278576226795846652382101
2907354897182427562197295231552018137414565442749272241125960796722557152453591693304764202
5814709794364855124394590463104036274829130885498544482251921593445114304907183386609528405
1162941958872971024878918092620807254965826177099708896450384318689022860981436677321905681
2325883917745942049757836185241614509931652354199417792900768637378045721962873354643811362
4651767835491884099515672370483229019863304708398835585801537274756091443925746709287622724
9303535670983768199031344740966458039726609416797671171603074549512182887851493418575245449
1860707134196753639806268948193291607945321883359534234320614909902436577570298683715049089
3721414268393507279612537896386583215890643766719068468641229819804873155140597367430098179
7442828536787014559225075792773166431781287533438136937282459639609746310281194734860196359
1488565707357402911845015158554633286356257506687627387456491927921949262056238946972039271
2977131414714805823690030317109266572712515013375254774912983855843898524112477893944078543
5954262829429611647380060634218533145425030026750509549825967711687797048224955787888157087
1190852565885922329476012126843706629085006005350101909965193542337559409644991157577631417
2381705131771844658952024253687413258170012010700203819930387084675118819289982315155262834
4763410263543689317904085073748265163400240214004076398607741693502376385799646303105256695
9526820527087378635808097014749653032680048042800815279721548338700475277159929260621051339
1905364105417475727161619402949930606536009608560163055944309667740095055431985852124210267
3810728210834951454323238805899861213072019217120326111888619335480190110863971704248420535
```

```
7621456421669902908646477611799722426144038434240652223777238670960380221727943408496841071
1524291284333980581729295522359944485228807686848130444755477341920760443455886816993682146
3048582568667961163458591044719888970457615373696260889510895468384152088691177363398736428
6097165137335922326917182089439777940915230747392521779021790936768304177382354726797472851
1219433027467184465383436417887955588183046149478504355804358187353660835476470945359494571
2438866054934368930766872835775911176366092298957008711608716374707321670952941890718989143
4877732109868737861533745671551822352732184597914017423217432749414643341905883781437978286
9755464219737475723067491343103644705464369195828034846434865498829286683811767562875956572
1951092843947495144613498268620728941092873839165606969286730997658573367623535125751913145
3902185687894990289226996537241457882185747678331213938573946199531714673524707025150382628
7804371375789980578453993074482915764371495356662427877147892399063429347049414050300765257
1560874275157996115690798614896583152874299071332485575429578479812685869409882810060153051
3121748550315992231381597229793166305748598142664971150859156959625371738819765620120306103
6243497100631984462763194459586332611497196285329942301718313919250743477639531240240612200
1248699420126396892552638891917266522299439257065988460343662783850148695527906248048122441
2497398840252793785105277783834533044598878514131976920687325567700297391055812496096244882
4994797680505587570210555567669066608919775702826395384137465113540059478211162499219248976
9989595361011175140421111135338132178395514056527907682749302270801189564223249984384979529
1997919072202235028084222270676264356791028113055815365498604541602379128446499968769959905
3995838144404470056168444454135252871358205622611163073099720908320475825689299993753991811
7991676288808940112336889082705057427164112452223261461994418166409516513785999875079836232
1598335257761788022467377781654101148543282249044465229239888363328190330275719997501596724
3196670515523576044934755563308202297086564498088930458479767266563806605514399950031934491
6393341031047152089869511126616404594173128996177860916959553453312761321102879990006386899
1278668206209430417973902225323280918834625799235572183391910690662552264220575998001277379
2557336412418860835947804450646561837669251598471144366783821381325104528441151996002554759
5114672824837721671895608901293123675338503196942288733567642762650209056882303992005109519
1022934564967544334379121780258624735067700639388457746713528552530041811376460798401021903
2045869129935088668758243560517249470135401278776915493427057105060083622752921596802043807
4091738259870177337516487121034498940270802557553830986854114210120167245505843193604087615
8183476519740354675032974242068997880541605115107661973708228420240334491011686387208175230
1636695303948070935006594848413799576108321023021532394741645684048066898202337277441635040
```

13.4 Markdown Cells

Text can be added to Jupyter Notebooks using Markdown cells. Markdown is a popular markup language that is a superset of HTML. Its specification can be found here:

http://daringfireball.net/projects/markdown/

13.4.1 Markdown basics

You can make text *italic* or **bold**.

You can build nested itemized or enumerated lists:

- One
  - Sublist
    * This
- Sublist - That - The other thing
- Two

- Sublist
- Three
- Sublist

Now another list:

1. Here we go

    (a) Sublist

    (b) Sublist

2. There we go

3. Now this

You can add horizontal rules:

Here is a blockquote:

> Beautiful is better than ugly. Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts. Special cases aren't special enough to break the rules. Although practicality beats purity. Errors should never pass silently. Unless explicitly silenced. In the face of ambiguity, refuse the temptation to guess. There should be one– and preferably only one –obvious way to do it. Although that way may not be obvious at first unless you're Dutch. Now is better than never. Although never is often better than *right* now. If the implementation is hard to explain, it's a bad idea. If the implementation is easy to explain, it may be a good idea. Namespaces are one honking great idea – let's do more of those!

And shorthand for links:

Jupyter's website

13.4.2 Headings

You can add headings by starting a line with one (or multiple) # followed by a space, as in the following example:

```
# Heading 1
# Heading 2
## Heading 2.1
## Heading 2.2
```

13.4.3 Embedded code

You can embed code meant for illustration instead of execution in Python:

```
def f(x):
    """a docstring"""
    return x**2
```

or other languages:

```
if (i=0; i<n; i++) {
  printf("hello %d\n", i);
  x += 4;
}
```

13.4.4 LaTeX equations

Courtesy of MathJax, you can include mathematical expressions both inline: $e^{i\pi} + 1 = 0$ and displayed:

$$e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i$$

Inline expressions can be added by surrounding the latex code with $:

```
$e^{i\pi} + 1 = 0$
```

Expressions on their own line are surrounded by $$:

```
$$e^x=\sum_{i=0}^\infty \frac{1}{i!}x^i$$
```

13.4.5 GitHub flavored markdown

The Notebook webapp supports Github flavored markdown meaning that you can use triple backticks for code blocks:

```
<pre>
```python
print "Hello World"
```
</pre>

<pre>
```javascript
console.log("Hello World")
```
</pre>
```

Gives:

```python
print "Hello World"
```

```
console.log("Hello World")
```

And a table like this:

```
<pre>
```

This	is
a	table

```
</pre>
```

A nice HTML Table:

This	is
a	table

13.4.6 General HTML

Because Markdown is a superset of HTML you can even add things like HTML tables:

Header 1

Header 2

row 1, cell 1

row 1, cell 2

row 2, cell 1

row 2, cell 2

13.4.7 Local files

If you have local files in your Notebook directory, you can refer to these files in Markdown cells directly:

```
[subdirectory/]<filename>
```

For example, in the images folder, we have the Python logo:

```
<img src="../images/python_logo.svg" />
```

and a video with the HTML5 video tag:

```
<video controls src="../images/animation.m4v" />
```

These do not embed the data into the notebook file, and require that the files exist when you are viewing the notebook.

Security of local files

Note that this means that the Jupyter notebook server also acts as a generic file server for files inside the same tree as your notebooks. Access is not granted outside the notebook folder so you have strict control over what files are visible, but for this reason it is highly recommended that you do not run the notebook server with a notebook directory at a high level in your filesystem (e.g. your home directory).

When you run the notebook in a password-protected manner, local file access is restricted to authenticated users unless read-only views are active.

13.5 Keyboard Shortcut Customization

Starting with IPython 2.0 keyboard shortcuts in command and edit mode are fully customizable. These customizations are made using the Jupyter JavaScript API. Here is an example that makes the r key available for running a cell:

```
In [1]: %%javascript

        Jupyter.keyboard_manager.command_shortcuts.add_shortcut('r', {
            help : 'run cell',
            help_index : 'zz',
            handler : function (event) {
                IPython.notebook.execute_cell();
                return false;
            }}
        );

<IPython.core.display.Javascript object>
```

"By default the keypress `r`, while in command mode, changes the type of the selected cell to `raw`. This shortcut is overridden by the code in the previous cell, and thus the action no longer be available via the keypress `r`."

There are a couple of points to mention about this API:

- The `help_index` field is used to sort the shortcuts in the Keyboard Shortcuts help dialog. It defaults to `zz`.

- When a handler returns `false` it indicates that the event should stop propagating and the default action should not be performed. For further details about the `event` object or event handling, see the jQuery docs.

- If you don't need a `help` or `help_index` field, you can simply pass a function as the second argument to `add_shortcut`.

```
In [2]: %%javascript

        Jupyter.keyboard_manager.command_shortcuts.add_shortcut('r', function (event) {
            IPython.notebook.execute_cell();
            return false;
        });
```

`<IPython.core.display.Javascript object>`

Likewise, to remove a shortcut, use `remove_shortcut`:

```
In [3]: %%javascript

        Jupyter.keyboard_manager.command_shortcuts.remove_shortcut('r');
```

`<IPython.core.display.Javascript object>`

If you want your keyboard shortcuts to be active for all of your notebooks, put the above API calls into your `custom.js` file.

Of course we provide name for majority of existing action so that you do not have to re-write everything, here is for example how to bind `r` back to it's initial behavior:

```
In [4]: %%javascript

        Jupyter.keyboard_manager.command_shortcuts.add_shortcut('r', 'jupyter-notebook:char
```

`<IPython.core.display.Javascript object>`

13.6 Embracing web standards

One of the main reasons why we developed the current notebook web application was to embrace the web technology.

By being a pure web application using HTML, Javascript, and CSS, the Notebook can get all the web technology improvement for free. Thus, as browser support for different media extend, the notebook web app should be able to be compatible without modification.

This is also true with performance of the User Interface as the speed of Javascript VM increases.

The other advantage of using only web technology is that the code of the interface is fully accessible to the end user and is modifiable live. Even if this task is not always easy, we strive to keep our code as accessible and reusable as possible. This should allow us - with minimum effort - development of small extensions that customize the behavior of the web interface.

13.6.1 Tampering with the Notebook application

The first tool that is available to you and that you should be aware of are browser "developers tool". The exact naming can change across browser and might require the installation of extensions. But basically they can allow you to inspect/modify the DOM, and interact with the javascript code that runs the frontend.

- In Chrome and Safari, Developer tools are in the menu `View > Developer > Javascript Console`
- In Firefox you might need to install Firebug

Those will be your best friends to debug and try different approaches for your extensions.

Injecting JS

Using magics

The above tools can be tedious for editing edit long JavaScript files. Therefore we provide the `%%javascript` magic. This allows you to quickly inject JavaScript into the notebook. Still the javascript injected this way will not survive reloading. Hence, it is a good tool for testing an refining a script.

You might see here and there people modifying css and injecting js into the notebook by reading file(s) and publishing them into the notebook. Not only does this often break the flow of the notebook and make the re-execution of the notebook broken, but it also means that you need to execute those cells in the entire notebook every time you need to update the code.

This can still be useful in some cases, like the `%autosave` magic that allows you to control the time between each save. But this can be replaced by a JavaScript dropdown menu to select the save interval.

```
In [1]: ## you can inspect the autosave code to see what it does.
        %autosave??
```

custom.js

To inject Javascript we provide an entry point: `custom.js` that allows the user to execute and load other resources into the notebook. Javascript code in `custom.js` will be executed when the notebook app starts and can then be used to customize almost anything in the UI and in the behavior of the notebook.

`custom.js` can be found in the Jupyter dir. You can share your custom.js with others.

Back to theory

```
In [2]: from jupyter_core.paths import jupyter_config_dir
        jupyter_dir = jupyter_config_dir()
        jupyter_dir
```

```
Out[2]: '/home/docs/.jupyter'
```

and custom js is in

```
In [3]: import os.path
        custom_js_path = os.path.join(jupyter_dir, 'custom', 'custom.js')
```

```
In [4]: #  my custom js
        if os.path.isfile(custom_js_path):
            with open(custom_js_path) as f:
                print(f.read())
```

```python
    else:
        print("You don't have a custom.js file")
```

```
You don't have a custom.js file
```

Note that `custom.js` is meant to be modified by user. When writing a script, you can define it in a separate file and add a line of configuration into `custom.js` that will fetch and execute the file.

Warning : even if modification of `custom.js` takes effect immediately after browser refresh (except if browser cache is aggressive), *creating* a file in `static/` directory needs a **server restart**.

13.6.2 Exercise :

- Create a `custom.js` in the right location with the following content:

```javascript
    alert("hello world from custom.js")
```

- Restart your server and open any notebook.

- Be greeted by custom.js

Have a look at default custom.js, to see it's content and for more explanation.

For the quick ones :

We've seen above that you can change the autosave rate by using a magic. This is typically something I don't want to type every time, and that I don't like to embed into my workflow and documents. (readers don't care what my autosave time is). Let's build an extension that allows us to do it.

Create a dropdown element in the toolbar (DOM `Jupyter.toolbar.element`), you will need

- `Jupyter.notebook.set_autosave_interval(miliseconds)`

- know that 1 min = 60 sec, and 1 sec = 1000 ms

```javascript
var label = jQuery('<label/>').text('AutoScroll Limit:');
var select = jQuery('<select/>')
    //.append(jQuery('<option/>').attr('value', '2').text('2min (default)'))
    .append(jQuery('<option/>').attr('value', undefined).text('disabled'))

    // TODO:
    //the_toolbar_element.append(label)
    //the_toolbar_element.append(select);

select.change(function() {
    var val = jQuery(this).val() // val will be the value in [2]
    // TODO
    // this will be called when dropdown changes

});

var time_m = [1,5,10,15,30];
for (var i=0; i < time_m.length; i++) {
    var ts = time_m[i];
                                    //[2]   ____ this will be `val` on [1]
                                    //       |
                                    //       v
    select.append($('<option/>').attr('value', ts).text(thr+'min'));
    // this will fill up the dropdown `select` with
```

```
    // 1 min
    // 5 min
    // 10 min
    // 10 min
    // ...
}
```

A non-interactive example first

I like my cython to be nicely highlighted

```
Jupyter.config.cell_magic_highlight['magic_text/x-cython'] = {}
Jupyter.config.cell_magic_highlight['magic_text/x-cython'].reg = [/^%%cython/]
```

`text/x-cython` is the name of CodeMirror mode name, `magic_` prefix will just patch the mode so that the first line that contains a magic does not screw up the highlighting. `reg` is a list or regular expression that will trigger the change of mode.

Get more documentation

Sadly, you will have to read the js source file (but there are lots of comments) and/or build the JavaScript documentation using yuidoc. If you have `node` and `yui-doc` installed:

```
$ cd ~/jupyter/notebook/notebook/static/notebook/js/
$ yuidoc . --server
warn: (yuidoc): Failed to extract port, setting to the default :3000
info: (yuidoc): Starting YUIDoc@0.3.45 using YUI@3.9.1 with NodeJS@0.10.15
info: (yuidoc): Scanning for yuidoc.json file.
info: (yuidoc): Starting YUIDoc with the following options:
info: (yuidoc):
{ port: 3000,
  nocode: false,
  paths: [ '.' ],
  server: true,
  outdir: './out' }
info: (yuidoc): Scanning for yuidoc.json file.
info: (server): Starting server: http://127.0.0.1:3000
```

and browse http://127.0.0.1:3000 to get documentation

Some convenience methods

By browsing the documentation you will see that we have some convenience methods that allows us to avoid re-inventing the UI every time :

```
Jupyter.toolbar.add_buttons_group([
        {
            'label'   : 'run qtconsole',
            'icon'    : 'icon-terminal', // select your icon from
                                         // http://fortawesome.github.io/Font-Awesome/icons/
            'callback': function(){Jupyter.notebook.kernel.execute('%qtconsole')}
        }
        // add more button here if needed.
        ]);
```

with a lot of icons you can select from.

13.6.3 Cell Metadata

The most requested feature is generally to be able to distinguish an individual cell in the notebook, or run a specific action with them. To do so, you can either use `Jupyter.notebook.get_selected_cell()`, or rely on `CellToolbar`. This allows you to register a set of actions and graphical elements that will be attached to individual cells.

Cell Toolbar

You can see some example of what can be done by toggling the `Cell Toolbar` selector in the toolbar on top of the notebook. It provides two default `presets` that are `Default` and `slideshow`. Default allows the user to edit the metadata attached to each cell manually.

First we define a function that takes at first parameter an element on the DOM in which to inject UI element. The second element is the cell this element wis registered with. Then we will need to register that function and give it a name.

Register a callback

```javascript
In [5]: %%javascript
        var CellToolbar = Jupyter.CellToolbar
        var toggle =  function(div, cell) {
            var button_container = $(div)

            // let's create a button that shows the current value of the metadata
            var button = $('<button/>').addClass('btn btn-mini').text(String(cell.metadata

            // On click, change the metadata value and update the button label
            button.click(function(){
                    var v = cell.metadata.foo;
                    cell.metadata.foo = !v;
                    button.text(String(!v));
                })

            // add the button to the DOM div.
            button_container.append(button);
        }

         // now we register the callback under the name foo to give the
         // user the ability to use it later
         CellToolbar.register_callback('tuto.foo', toggle);
<IPython.core.display.Javascript object>
```

Registering a preset

This function can now be part of many `preset` of the CellToolBar.

```
In [6]: %%javascript
        Jupyter.CellToolbar.register_preset('Tutorial 1',['tuto.foo','default.rawedit'])
        Jupyter.CellToolbar.register_preset('Tutorial 2',['slideshow.select','tuto.foo'])
```

```
<IPython.core.display.Javascript object>
```

You should now have access to two presets :

- Tutorial 1

- Tutorial 2

And check that the buttons you defined share state when you toggle preset. Also check that the metadata of the cell is modified when you click the button, and that when saved on reloaded the metadata is still available.

Exercise:

Try to wrap the all code in a file, put this file in `{jupyter_dir}/custom/<a-name>.js`, and add

```
require(['custom/<a-name>']);
```

in `custom.js` to have this script automatically loaded in all your notebooks.

`require` is provided by a javascript library that allow you to express dependency. For simple extension like the previous one we directly mute the global namespace, but for more complex extension you could pass a callback to `require([...], <callback>)` call, to allow the user to pass configuration information to your plugin.

In Python lang,

```
require(['a/b', 'c/d'], function( e, f){
    e.something()
    f.something()
})
```

could be read as

```
import a.b as e
import c.d as f
e.something()
f.something()
```

See for example @damianavila ["ZenMode" plugin](https://github.com/ipython-contrib/IPython-notebook-extensions/blob/master/custom.example.js#L34) :

```
// read that as
// import custom.zenmode.main as zenmode
require(['custom/zenmode/main'],function(zenmode){
    zenmode.background('images/back12.jpg');
})
```

For the quickest

Try to use the following to bind a dropdown list to `cell.metadata.difficulty.select`.

It should be able to take the 4 following values :

- `<None>`

- `Easy`

- Medium
- Hard

We will use it to customize the output of the converted notebook depending on the tag on each cell

```
In [7]: # %load soln/celldiff.js
```

```
In [8]:
```

13.7 Importing Jupyter Notebooks as Modules

It is a common problem that people want to import code from Jupyter Notebooks. This is made difficult by the fact that Notebooks are not plain Python files, and thus cannot be imported by the regular Python machinery.

Fortunately, Python provides some fairly sophisticated hooks into the import machinery, so we can actually make Jupyter notebooks importable without much difficulty, and only using public APIs.

```
In [1]: import io, os, sys, types
```

```
In [2]: from IPython import get_ipython
        from IPython.nbformat import current
        from IPython.core.interactiveshell import InteractiveShell
```

```
/home/docs/checkouts/readthedocs.org/user_builds/test-notebook/conda/doc-config/lib/python3
  "You should import from nbformat instead.", ShimWarning)
/home/docs/checkouts/readthedocs.org/user_builds/test-notebook/conda/doc-config/lib/python3

- use nbformat for read/write/validate public API
- use nbformat.vX directly to composing notebooks of a particular version

    """)
```

Import hooks typically take the form of two objects:

1. a Module **Loader**, which takes a module name (e.g. 'IPython.display'), and returns a Module
2. a Module **Finder**, which figures out whether a module might exist, and tells Python what **Loader** to use

```
In [3]: def find_notebook(fullname, path=None):
            """find a notebook, given its fully qualified name and an optional path

            This turns "foo.bar" into "foo/bar.ipynb"
            and tries turning "Foo_Bar" into "Foo Bar" if Foo_Bar
            does not exist.
            """
            name = fullname.rsplit('.', 1)[-1]
            if not path:
                path = ['']
            for d in path:
                nb_path = os.path.join(d, name + ".ipynb")
                if os.path.isfile(nb_path):
                    return nb_path
                # let import Notebook_Name find "Notebook Name.ipynb"
                nb_path = nb_path.replace("_", " ")
                if os.path.isfile(nb_path):
```

```
                   return nb_path
```

13.7.1 Notebook Loader

Here we have our Notebook Loader. It's actually quite simple - once we figure out the filename of the module, all it does is:

1. load the notebook document into memory

2. create an empty Module

3. execute every cell in the Module namespace

Since IPython cells can have extended syntax, the IPython transform is applied to turn each of these cells into their pure-Python counterparts before executing them. If all of your notebook cells are pure-Python, this step is unnecessary.

```python
In [4]: class NotebookLoader(object):
            """Module Loader for Jupyter Notebooks"""
            def __init__(self, path=None):
                self.shell = InteractiveShell.instance()
                self.path = path

            def load_module(self, fullname):
                """import a notebook as a module"""
                path = find_notebook(fullname, self.path)

                print ("importing Jupyter notebook from %s" % path)

                # load the notebook object
                with io.open(path, 'r', encoding='utf-8') as f:
                    nb = current.read(f, 'json')


                # create the module and add it to sys.modules
                # if name in sys.modules:
                #    return sys.modules[name]
                mod = types.ModuleType(fullname)
                mod.__file__ = path
                mod.__loader__ = self
                mod.__dict__['get_ipython'] = get_ipython
                sys.modules[fullname] = mod

                # extra work to ensure that magics that would affect the user_ns
                # actually affect the notebook module's ns
                save_user_ns = self.shell.user_ns
                self.shell.user_ns = mod.__dict__

                try:
                  for cell in nb.worksheets[0].cells:
                    if cell.cell_type == 'code' and cell.language == 'python':
                        # transform the input to executable Python
                        code = self.shell.input_transformer_manager.transform_cell(cell.inp
                        # run the code in themodule
                        exec(code, mod.__dict__)
                finally:
```

```
                self.shell.user_ns = save_user_ns
            return mod
```

13.7.2 The Module Finder

The finder is a simple object that tells you whether a name can be imported, and returns the appropriate loader. All this one does is check, when you do:

```python
import mynotebook
```

it checks whether `mynotebook.ipynb` exists. If a notebook is found, then it returns a NotebookLoader.

Any extra logic is just for resolving paths within packages.

```python
In [5]: class NotebookFinder(object):
            """Module finder that locates Jupyter Notebooks"""
            def __init__(self):
                self.loaders = {}

            def find_module(self, fullname, path=None):
                nb_path = find_notebook(fullname, path)
                if not nb_path:
                    return

                key = path
                if path:
                    # lists aren't hashable
                    key = os.path.sep.join(path)

                if key not in self.loaders:
                    self.loaders[key] = NotebookLoader(path)
                return self.loaders[key]
```

13.7.3 Register the hook

Now we register the `NotebookFinder` with `sys.meta_path`

```python
In [6]: sys.meta_path.append(NotebookFinder())
```

After this point, my notebooks should be importable.

Let's look at what we have in the CWD:

```python
In [7]: ls nbpackage
```

`__init__.py` `__pycache__/` `mynotebook.ipynb` `nbs/`

So I should be able to `import nbimp.mynotebook`.

Aside: displaying notebooks

Here is some simple code to display the contents of a notebook with syntax highlighting, etc.

```
In [8]: from pygments import highlight
        from pygments.lexers import PythonLexer
        from pygments.formatters import HtmlFormatter

        from IPython.display import display, HTML

        formatter = HtmlFormatter()
        lexer = PythonLexer()

        # publish the CSS for pygments highlighting
        display(HTML("""
        <style type='text/css'>
        %s
        </style>
        """ % formatter.get_style_defs()
        ))
In [9]: def show_notebook(fname):
            """display a short summary of the cells of a notebook"""
            with io.open(fname, 'r', encoding='utf-8') as f:
                nb = current.read(f, 'json')
            html = []
            for cell in nb.worksheets[0].cells:
                html.append("<h4>%s cell</h4>" % cell.cell_type)
                if cell.cell_type == 'code':
                    html.append(highlight(cell.input, lexer, formatter))
                else:
                    html.append("<pre>%s</pre>" % cell.source)
            display(HTML('\n'.join(html)))

        show_notebook(os.path.join("nbpackage", "mynotebook.ipynb"))
```

So my notebook has a heading cell and some code cells, one of which contains some IPython syntax.

Let's see what happens when we import it

```
In [10]: from nbpackage import mynotebook
```

importing Jupyter notebook from /home/docs/checkouts/readthedocs.org/user_builds/test-noteb

Hooray, it imported! Does it work?

```
In [11]: mynotebook.foo()
```

```
Out[11]: 'foo'
```

Hooray again!

Even the function that contains IPython syntax works:

```
In [12]: mynotebook.has_ip_syntax()
```

```
Out[12]: ['Connecting with the Qt Console.ipynb',
         'Custom Keyboard Shortcuts.ipynb',
         'Distributing Jupyter Extensions as Python Packages.ipynb',
         'Importing Notebooks.ipynb',
         'JavaScript Notebook Extensions.ipynb',
         'Notebook Basics.ipynb',
         'Running Code.ipynb',
         'Typesetting Equations.ipynb',
```

```
'What is the Jupyter Notebook.ipynb',
'Working With Markdown Cells.ipynb',
'examples_index.rst',
'images',
'nbpackage']
```

13.7.4 Notebooks in packages

We also have a notebook inside the `nb` package, so let's make sure that works as well.

In [13]: ls nbpackage/nbs

__init__.py __pycache__/ other.ipynb

Note that the `__init__.py` is necessary for `nb` to be considered a package, just like usual.

In [14]: show_notebook(os.path.join("nbpackage", "nbs", "other.ipynb"))

In [15]: **from nbpackage.nbs import** other
        other.bar(5)

importing Jupyter notebook from /home/docs/checkouts/readthedocs.org/user_builds/test-noteb

Out[15]: 'barbarbarbarbar'

So now we have importable notebooks, from both the local directory and inside packages.

I can even put a notebook inside IPython, to further demonstrate that this is working properly:

In [16]: **import shutil**
        from IPython.utils.path import get_ipython_package_dir

        utils = os.path.join(get_ipython_package_dir(), 'utils')
        shutil.copy(os.path.join("nbpackage", "mynotebook.ipynb"),
                    os.path.join(utils, "inside_ipython.ipynb")
)

/home/docs/checkouts/readthedocs.org/user_builds/test-notebook/conda/doc-config/lib/python3
  warn("get_ipython_package_dir has moved to the IPython.paths module")

Out[16]: '/home/docs/checkouts/readthedocs.org/user_builds/test-notebook/conda/doc-config/l

and import the notebook from `IPython.utils`

In [17]: **from IPython.utils import** inside_ipython
        inside_ipython.whatsmyname()

importing Jupyter notebook from /home/docs/checkouts/readthedocs.org/user_builds/test-noteb

Out[17]: 'IPython.utils.inside_ipython'

This approach can even import functions and classes that are defined in a notebook using the `%%cython` magic.

13.8 Connecting to an existing IPython kernel using the Qt Console

13.8.1 The Frontend/Kernel Model

The traditional IPython (`ipython`) consists of a single process that combines a terminal based UI with the process that runs the users code.

While this traditional application still exists, the modern Jupyter consists of two processes:

- Kernel: this is the process that runs the users code.

- Frontend: this is the process that provides the user interface where the user types code and sees results.

Jupyter currently has 3 frontends:

- Terminal Console (`ipython console`)

- Qt Console (`ipython qtconsole`)

- Notebook (`ipython notebook`)

The Kernel and Frontend communicate over a ZeroMQ/JSON based messaging protocol, which allows multiple Frontends (even of different types) to communicate with a single Kernel. This opens the door for all sorts of interesting things, such as connecting a Console or Qt Console to a Notebook's Kernel. For example, you may want to connect a Qt console to your Notebook's Kernel and use it as a help browser, calling `??` on objects in the Qt console (whose pager is more flexible than the one in the notebook).

This Notebook describes how you would connect another Frontend to a Kernel that is associated with a Notebook.

13.8.2 Manual connection

To connect another Frontend to a Kernel manually, you first need to find out the connection information for the Kernel using the `%connect_info` magic:

```
In [1]: %connect_info

  "control_port": 38452,  "iopub_port": 56114,  "key": "89d440a2-0299-4654-ad3e-029dcaff20:

Paste the above JSON into a file, and connect with:
    $> ipython <app> --existing <file>
or, if you are local, you can connect with just:
    $> ipython <app> --existing /tmp/tmp9l2j3w2x.json
or even just:
    $> ipython <app> --existing
if this is the most recent IPython session you have started.
```

You can see that this magic displays everything you need to connect to this Notebook's Kernel.

13.8.3 Automatic connection using a new Qt Console

You can also start a new Qt Console connected to your current Kernel by using the `%qtconsole` magic. This will detect the necessary connection information and start the Qt Console for you automatically.

```
In [2]: a = 10
```

```
In [3]: %qtconsole
```

The Markdown parser included in the Jupyter Notebook is MathJax-aware. This means that you can freely mix in mathematical expressions using the MathJax subset of Tex and LaTeX. Some examples from the MathJax site are reproduced below, as well as the Markdown+TeX source.

13.9 Motivating Examples

13.9.1 The Lorenz Equations

Source

```
\begin{align}
\dot{x} & = \sigma(y-x) \\
\dot{y} & = \rho x - y - xz \\
\dot{z} & = -\beta z + xy
\end{align}
```

Display

13.9.2 The Cauchy-Schwarz Inequality

Source

```
\begin{equation*}
\left( \sum_{k=1}^n a_k b_k \right)^2 \leq \left( \sum_{k=1}^n a_k^2 \right) \left( \sum_{k=1}^n b_k^
\end{equation*}
```

Display

$$\left(\sum_{k=1}^n a_k b_k \right)^2 \leq \left(\sum_{k=1}^n a_k^2 \right) \left(\sum_{k=1}^n b_k^2 \right)$$

13.9.3 A Cross Product Formula

Source

```
\begin{equation*}
\mathbf{V}_1 \times \mathbf{V}_2 =  \begin{vmatrix}
\mathbf{i} & \mathbf{j} & \mathbf{k} \\
\frac{\partial X}{\partial u} &  \frac{\partial Y}{\partial u} & 0 \\
\frac{\partial X}{\partial v} &  \frac{\partial Y}{\partial v} & 0
\end{vmatrix}
\end{equation*}
```

Display

$$\mathbf{V}_1 \times \mathbf{V}_2 = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ \frac{\partial X}{\partial u} & \frac{\partial Y}{\partial u} & 0 \\ \frac{\partial X}{\partial v} & \frac{\partial Y}{\partial v} & 0 \end{vmatrix}$$

13.9.4 The probability of getting (k) heads when flipping (n) coins is

Source

```
\begin{equation*}
P(E)   = {n \choose k} p^k (1-p)^{ n-k}
\end{equation*}
```

Display

$$P(E) = \binom{n}{k} p^k (1-p)^{n-k}$$

13.9.5 An Identity of Ramanujan

Source

```
\begin{equation*}
\frac{1}{\Bigl(\sqrt{\phi \sqrt{5}}-\phi\Bigr) e^{\frac25 \pi}} =
1+\frac{e^{-2\pi}} {1+\frac{e^{-4\pi}} {1+\frac{e^{-6\pi}}
{1+\frac{e^{-8\pi}} {1+\ldots} } } }
\end{equation*}
```

Display

$$\frac{1}{\left(\sqrt{\phi\sqrt{5}}-\phi\right)e^{\frac{2}{5}\pi}} = 1 + \cfrac{e^{-2\pi}}{1+\cfrac{e^{-4\pi}}{1+\cfrac{e^{-6\pi}}{1+\cfrac{e^{-8\pi}}{1+\ldots}}}}$$

13.9.6 A Rogers-Ramanujan Identity

Source

```
\begin{equation*}
1 +  \frac{q^2}{(1-q)}+\frac{q^6}{(1-q)(1-q^2)}+\cdots =
\prod_{j=0}^{\infty}\frac{1}{(1-q^{5j+2})(1-q^{5j+3})},
\quad\quad \text{for $|q|<1$}.
\end{equation*}
```

Display

$$1+\frac{q^2}{(1-q)}+\frac{q^6}{(1-q)(1-q^2)}+\cdots = \prod_{j=0}^{\infty}\frac{1}{(1-q^{5j+2})(1-q^{5j+3})}, \qquad \text{for } |q| < 1.$$

13.9.7 Maxwell's Equations

Source

```
\begin{align}
\nabla \times \vec{\mathbf{B}} -\, \frac1c\, \frac{\partial\vec{\mathbf{E}}}{\partial t} & = \frac{4
\nabla \times \vec{\mathbf{E}}\, +\, \frac1c\, \frac{\partial\vec{\mathbf{B}}}{\partial t} & = \vec{
\nabla \cdot \vec{\mathbf{B}} & = 0
\end{align}
```

Display

13.9.8 Equation Numbering and References

Equation numbering and referencing will be available in a future version of the Jupyter notebook.

13.9.9 Inline Typesetting (Mixing Markdown and TeX)

While display equations look good for a page of samples, the ability to mix math and *formatted* **text** in a paragraph is also important.

Source

```
This expression $\sqrt{3x-1}+(1+x)^2$ is an example of a TeX inline equation in a [Markdown-formatted
```

Display

This expression $\sqrt{3x-1}+(1+x)^2$ is an example of a TeX inline equation in a Markdown-formatted sentence.

13.9.10 Other Syntax

You will notice in other places on the web that `$$` are needed explicitly to begin and end MathJax typesetting. This is **not** required if you will be using TeX environments, but the Jupyter notebook will accept this syntax on legacy notebooks.

13.9.11 Source

```
$$
\begin{array}{c}
y_1 \\
y_2 \mathtt{t}_i \\
z_{3,4}
\end{array}
$$
```

```
$$
\begin{array}{c}
y_1 \cr
y_2 \mathtt{t}_i \cr
y_{3}
\end{array}
$$
```

```
$$\begin{eqnarray}
x' &=& &x \sin\phi &+& z \cos\phi \\
z' &=& - &x \cos\phi &+& z \sin\phi \\
\end{eqnarray}$$
```

```
$$
x=4
$$
```

13.9.12 Display

$$
\begin{array}{c}
y_1 \\
y_2 \mathtt{t}_i \\
z_{3,4}
\end{array}
$$

$$
\begin{array}{c}
y_1 \\
y_2 \mathtt{t}_i \\
y_3
\end{array}
$$

to

$$x' =$$

$$x \sin \phi$$

$$+z \cos \phi$$

$$z' =$$

$$-x \cos \phi$$

$$+z \sin \phi$$

(13.1)

$$=$$

$$+$$

$$- =$$

$$+x \cos \phi$$

$$z \sin \phi$$

(13.1)

$$x = 4$$

My Notebook

```
In [1]: def foo():
            return "foo"

In [2]: def has_ip_syntax():
            listing = !ls
            return listing

In [3]: def whatsmyname():
            return __name__
```

Other notebook

This notebook just defines `bar`

```
In [1]: def bar(x):
            return "bar" * x
```

Jupyter notebook changelog

A summary of changes in the Jupyter notebook. For more detailed information, see GitHub.

16.1 4.1.0

Bug fixes:

- Properly reap zombie subprocesses
- Fix cross-origin problems
- Fix double-escaping of the base URL prefix
- Handle invalid unicode filenames more gracefully
- Fix ANSI color-processing
- Send keepalive messages for web terminals
- Fix bugs in the notebook tour

UI changes:

- Moved the cell toolbar selector into the *View* menu. Added a button that triggers a "hint" animation to the main toolbar so users can find the new location. (Click here to see a screencast)
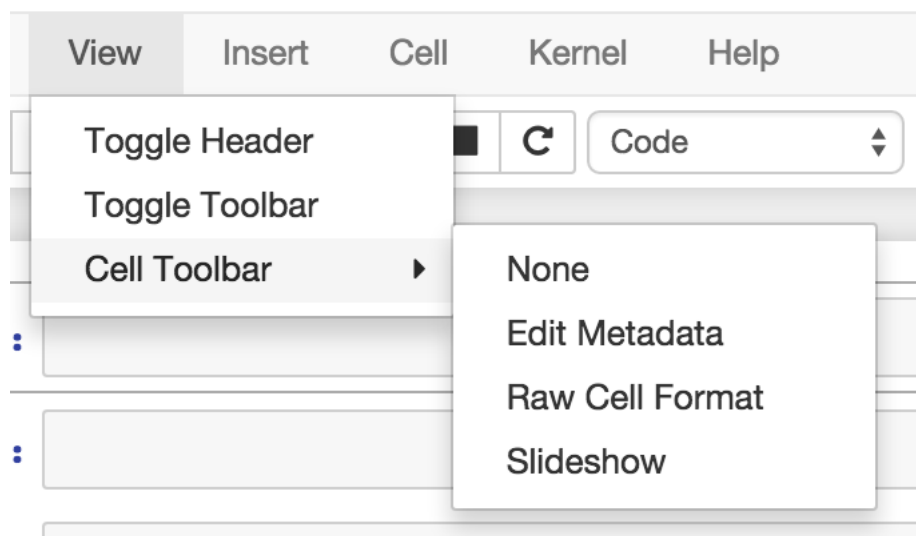
- Added *Restart & Run All* to the *Kernel* menu. Users can also bind it to a keyboard shortcut on action `restart-kernel-and-run-all-cells`.

- Added multiple-cell selection. Users press `Shift-Up/Down` or `Shift-K/J` to extend selection in command mode. Various actions such as cut/copy/paste, execute, and cell type conversions apply to all selected cells.

Code cells allow you to enter and run code

Run a code cell using `Shift-Enter` or pressing the ▶ button in the toolbar above:

```
In [ ]:  a = 10
```

```
In [ ]:  print(a)
```

There are two other keyboard shortcuts for running code:

- `Alt-Enter` runs the current cell and inserts a new one below.
- `Ctrl-Enter` run the current cell and enters command mode.

- Added a command palette for executing Jupyter actions by name. Users press `Cmd/Ctrl-Shift-P` or click the new command palette icon on the toolbar.

find and replace	🔍
jupyter-notebook command group	
find and replace	(command) F
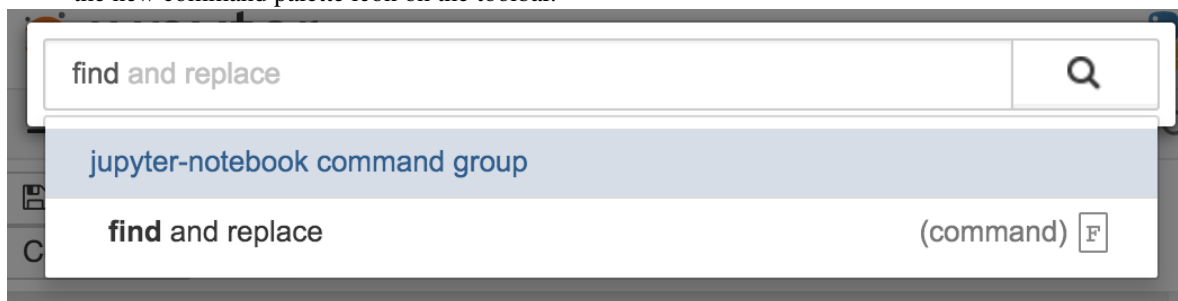
- Added a *Find and Replace* dialog to the *Edit* menu. Users can also press `F` in command mode to show the dialog.
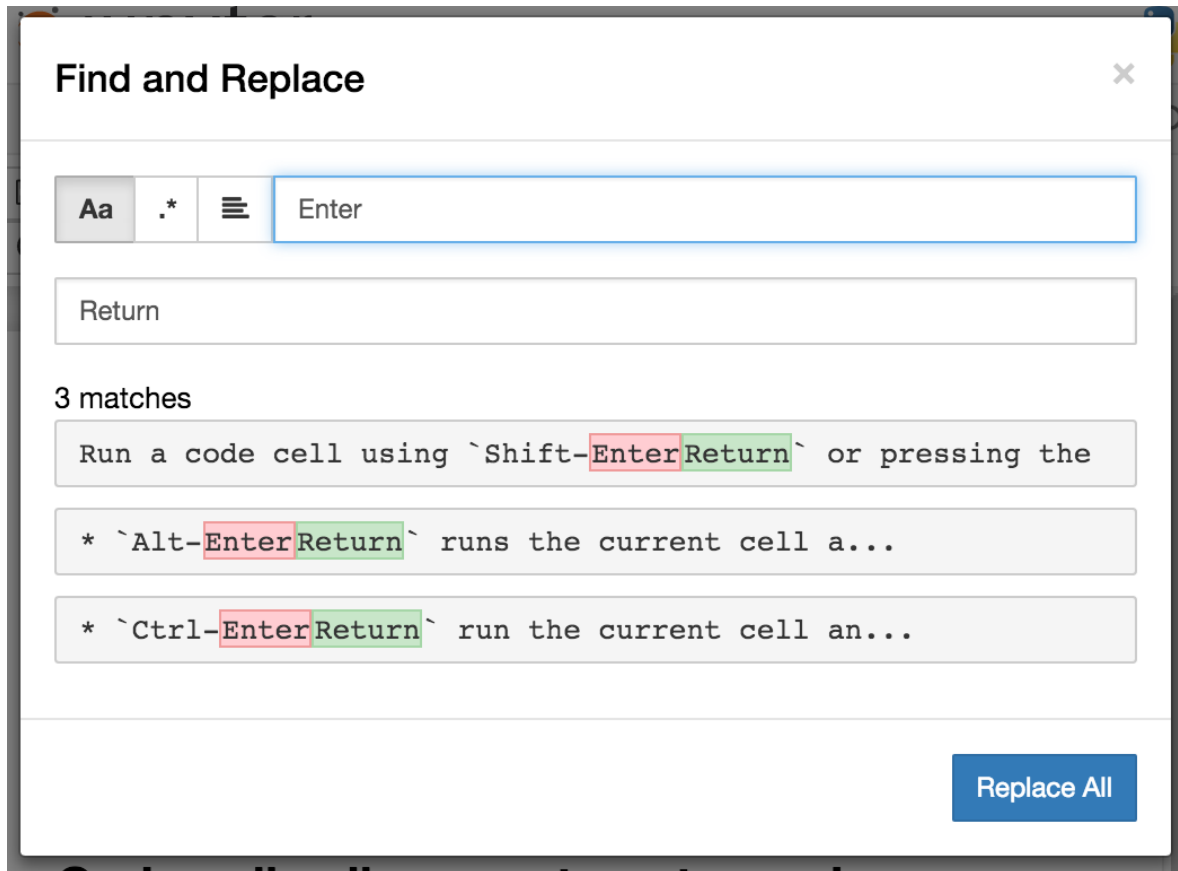
Other improvements:

- Custom KernelManager methods can be Tornado coroutines, allowing async operations.
- Make clearing output optional when rewriting input with `set_next_input(replace=True)`.
- Added support for TLS client authentication via `--NotebookApp.client-ca`.
- Added tags to `jupyter/notebook` releases on DockerHub. `latest` continues to track the master branch.

See the 4.1 milestone on GitHub for a complete list of issues and pull requests handled.

16.2 4.0.x

16.2.1 4.0.6

- fix installation of mathjax support files
- fix some double-escape regressions in 4.0.5
- fix a couple of cases where errors could prevent opening a notebook

16.2.2 4.0.5

Security fixes for maliciously crafted files.

- CVE-2015-6938: malicious filenames

- CVE-2015-7337: malicious binary files in text editor.

Thanks to Jonathan Kamens at Quantopian and Juan Broullón for the reports.

16.2.3 4.0.4

- Fix inclusion of mathjax-safe extension

16.2.4 4.0.2

- Fix launching the notebook on Windows
- Fix the path searched for frontend config

16.2.5 4.0.0

First release of the notebook as a standalone package.