

---

# **scrapbook Documentation**

***Release 0.5.0***

**nteract team**

**Apr 13, 2022**



# CONTENTS

<b>1</b>	<b>Python Version Support</b>	<b>3</b>
<b>2</b>	<b>Use Case</b>	<b>5</b>
<b>3</b>	<b>Documentation</b>	<b>7</b>
<b>4</b>	<b>API Reference</b>	<b>15</b>
<b>5</b>	<b>Indices and tables</b>	<b>27</b>
	<b>Python Module Index</b>	<b>29</b>
	<b>Index</b>	<b>31</b>



**scrapbook** is a library for recording a notebook’s data values and generated visual content as “scraps”. These recorded scraps can be read at a future time.

This library replaces [papermill](#)’s existing *record* functionality.



## PYTHON VERSION SUPPORT

This library supports python 3.6+.





## USE CASE

Notebook users may wish to record data produced during a notebook execution. This recorded data can then be read to be used at a later time or be passed to another notebook as input.

Namely scrapbook lets you:

- **persist** data and displays (scraps) in a notebook
- **recall** any persisted scrap of data
- **summarize collections** of notebooks



## DOCUMENTATION

These pages guide you through the installation and usage of scrapbook.

### 3.1 Installation

#### 3.1.1 Installing the application

From the command line:

```
pip install scrapbook
```

For all optional io dependencies, you can specify individual bundles like `s3`, or `azure` – or use `all`

```
pip install scrapbook[all]
```

### 3.2 Models

A few new names for information are introduced in scrapbook:

- **scraps**: serializable data values and visualizations such as strings, lists of objects, pandas dataframes, charts, images, or data references.
- **notebook**: a wrapped nbformat notebook object with extra methods for interacting with scraps.
- **scrapbook**: a collection of notebooks with an interface for asking questions of the collection.
- **encoders**: a registered translator of data to/from notebook storage formats.

#### 3.2.1 Scrap

The scrap model houses a few key attributes in a tuple. Namely:

- **name**: The name of the scrap
- **data**: Any data captured by the scrapbook api call
- **encoder**: The name of the encoder used to encode/decode data to/from the notebook
- **display**: Any display data used by IPython to display visual content

### 3.2.2 Notebook

The Notebook object adheres to the `nbformat`'s `json` schema, allowing for access to its required fields.

```
nb = sb.read_notebook('notebook.ipynb')
nb.cells # The cells from the notebook
nb.metadata
nb.nbformat
nb.nbformat_minor
```

There's a few additional methods provided, outlined in the API page ([read\\_notebook API](#))

### 3.2.3 Scrapbook

A collection of Notebooks is called a Scrapbook. It allows for access the underlying notebooks and to perform data collection from the group as a whole.

```
# create a scrapbook named `book`
book = sb.read_notebooks('path/to/notebook/collection/')
# get the underlying notebooks as a list
book.notebooks # Or `book.values`
```

There's a additional methods provided, outlined in the API page ([read\\_notebooks API](#))

### 3.2.4 Encoder

Encoders are accessible by key names to Encoder objects registered against the `encoders.registry` object. To register new data encoders simply call:

```
from scrapbook.encoders import registry as encoder_registry
# add encoder to the registry
encoder_registry.register("custom_encoder_name", MyCustomEncoder())
```

The encode class must implement two methods, `encode` and `decode`:

```
class MyCustomEncoder(object):
    def encode(self, scrap):
        # scrap.data is any type, usually specific to the encoder name
        pass # Return a `Scrap` with `data` type one of [None, list, dict, *six.
        ↪integer_types, *six.string_types]

    def decode(self, scrap):
        # scrap.data is one of [None, list, dict, *six.integer_types, *six.string_
        ↪types]
        pass # Return a `Scrap` with `data` type as any type, usually specific to
        ↪the encoder name
```

This can read transform scraps into a json object representing their contents or location and load those strings back into the original data objects.

For example, here is the code for a custom encoder that can save [Altair charts](#) by converting the chart to a dictionary as a part of the encoding process.

```

from scrapbook.encoders import registry as encoder_registry
import altair as alt

class AltairEncoder(object):
    def encode(self, scrap):
        # Here we assume the input to `sb.glue` is an Altair chart.
        scrap = scrap._replace(data=scrap.data.to_dict())
        return scrap

    def decode(self, scrap):
        scrap = scrap._replace(data=alt.Chart.from_dict(scrap.data))
        return scrap

# Register the encoder so that scrapbook can use it
encoder_registry.register("altair", AltairEncoder())
# Now we can use this encoder with `glue`
sb.glue('my_altair_chart', chart, 'altair')

```

## text

A basic string storage format that saves data as python strings.

```
sb.glue("hello", "world", "text")
```

## json

```
sb.glue("foo_json", {"foo": "bar", "baz": 1}, "json")
```

## arrow

Implementation Pending!

## 3.3 glue API

The glue call records a *Scrap* (data or display value) in the given notebook cell.

The scrap (recorded value) can be retrieved during later inspection of the output notebook.

```

import scrapbook as sb

sb.glue("hello", "world")
sb.glue("number", 123)
sb.glue("some_list", [1, 3, 5])
sb.glue("some_dict", {"a": 1, "b": 2})
sb.glue("non_json", df, 'pandas')

```

The scrapbook library can be used later to recover scraps (recorded values) from the output notebook:

```

nb = sb.read_notebook('notebook.ipynb')
nb.scraps

```

**scrapbook** will imply the storage format by the value type of any registered data encoders. Alternatively, the implied encoding format can be overwritten by setting the `encoder` argument to the registered name (e.g. `"json"`) of a particular encoder.

This data is persisted by generating a display output with a special media type identifying the content encoding format and data. These outputs are not always visible in notebook rendering but still exist in the document. Scrapbook can then rehydrate the data associated with the notebook in the future by reading these cell outputs.

### 3.3.1 Pandas

When glueing pandas dataframes, the library will use pyarrow to translate the dataframe to a base64 encoded parquet file. Because of this tool chain, certain nested objects will not encode cleanly and will raise an Arrow exception. Common nested objects that will fail include columns with dicts or sets within them, either directly or nested inside other objects. Over time these nested types should be more supported (nested lists work for example) as Arrow adds struct transformations.

### 3.3.2 Display Outputs

To display a named scrap with visible display outputs, you need to indicate that the scrap is directly renderable.

This can be done by toggling the `display` argument.

```
# record a UI message along with the input string
sb.glue("hello", "Hello World", display=True)
```

The call will save the data and the display attributes of the Scrap object, making it visible as well as encoding the original data. This leans on the `IPython.core.formatters.format_display_data` function to translate the data object into a display and metadata dict for the notebook kernel to parse.

Another pattern that can be used is to specify that **only the display data** should be saved, and not the original object. This is achieved by setting the encoder to be `display`.

```
# record an image without the original input object
sb.glue("sharable_png",
       IPython.display.Image(filename="sharable.png"),
       encoder='display'
)
```

Finally the media types that are generated can be controlled by passing a list, tuple, or dict object as the `display` argument.

```
sb.glue("media_as_text_only",
       media_obj,
       encoder='display',
       display=('text/plain',) # This passes [text/plain] to format_display_data's include_
       ↪ argument
)

sb.glue("media_without_text",
       media_obj,
       encoder='display',
       display={'exclude': 'text/plain'} # forward to format_display_data's kwargs
)
```

Like data scraps, these can be retrieved at a later time by accessing the scrap's `display` attribute. Though usually one will just use Notebook's `reglue` method (*reglue*).

### An example using display data

For example, the following code generates a Matplotlib plot and saves *only* the display data as a scrap. This allows you to import the plot into another notebook.

```
# Generate our plot
fig, ax = plt.subplots()
ax.plot(x, y)

# We use *fig* as IPython knows how to display this.
sb.glue("sharable_plot", fig, "display")
```

This glues *only* the display information (e.g. the base64 encoded image generated by Matplotlib). In another notebook, it can be accessed and displayed like so:

```
nb = sb.read_notebook(path_to_first_notebook)

# To display the image and reglue it
nb.reglue('sharable_plot')

# To access the display information directly
nb.scrap['sharable_plot'].display['data']['image/png']
```

## 3.4 read\_notebook API

Reads a *Notebook* object loaded from the location specified at `path`. You've already seen how this function is used in the above api call examples, but essentially this provides a thin wrapper over an `nbformat`'s `NotebookNode` with the ability to extract scrapbook scraps.

```
nb = sb.read_notebook('notebook.ipynb')
```

This Notebook object adheres to the `nbformat`'s `json schema`, allowing for access to its required fields.

```
nb.cells # The cells from the notebook
nb.metadata
nb.nbformat
nb.nbformat_minor
```

There's a few additional methods provided, most of which are outlined in more detail below:

```
nb.scrap
nb.reglue
```

The abstraction also makes saved content available as a dataframe referencing each key and source. More of these methods will be made available in later versions.

```
# Produces a data frame with ["name", "data", "encoder", "display", "filename"] as_
↳ columns
nb.scrap_dataframe # Warning: This might be a large object if data or display is large
```

The Notebook object also has a few legacy functions for backwards compatibility with papermill's Notebook object model. As a result, it can be used to read papermill execution statistics as well as scrapbook abstractions:

```
nb.cell_timing # List of cell execution timings in cell order
nb.execution_counts # List of cell execution counts in cell order
nb.papermill_metrics # Dataframe of cell execution counts and times
nb.papermill_record_dataframe # Dataframe of notebook records (scraps with only data)
nb.parameter_dataframe # Dataframe of notebook parameters
nb.papermill_dataframe # Dataframe of notebook parameters and cell scraps
```

The notebook reader relies on `papermill`'s registered `iowr` to enable access to a variety of sources such as – but not limited to – S3, Azure, and Google Cloud.

### 3.4.1 scraps

The `scraps` method allows for access to all of the scraps in a particular notebook by providing a name -> scrap lookup.

```
nb = sb.read_notebook('notebook.ipynb')
nb.scraps # Prints a dict of all scraps by name
```

This object has a few additional methods as well for convenient conversion and execution.

```
nb.scraps.data_scraps # Filters to only scraps with `data` associated
nb.scraps.data_dict # Maps `data_scraps` to a `name` -> `data` dict
nb.scraps.display_scraps # Filters to only scraps with `display` associated
nb.scraps.display_dict # Maps `display_scraps` to a `name` -> `display` dict
nb.scraps.dataframe # Generates a dataframe with ["name", "data", "encoder", "display
→ "] as columns
```

These methods allow for simple use-cases to not require digging through model abstractions.

### 3.4.2 reglue

Using `reglue` one can take any scrap glue'd into one notebook and glue into the current one.

```
nb = sb.read_notebook('notebook.ipynb')
nb.reglue("table_scrap") # This copies both data and displays
```

Any data or display information will be copied verbatim into the currently executing notebook as though the user called `glue` again on the original source.

It's also possible to rename the scrap in the process.

```
nb.reglue("table_scrap", "old_table_scrap")
```

And finally if one wishes to try to `reglue` without checking for existence the `raise_on_missing` can be set to just display a message on failure.

```
nb.reglue("maybe_missing", raise_on_missing=False)
# => "No scrap found with name 'maybe_missing' in this notebook"
```



## 3.5 read\_notebooks API

Reads all notebooks located in a given path into a *Scrapbook* object.

```
# create a scrapbook named `book`
book = sb.read_notebooks('path/to/notebook/collection/')
# get the underlying notebooks as a list
book.notebooks # Or `book.values`
```

The path reuses papermill's registered iorw. to list and read files from various sources, such that non-local urls can load data.

```
# create a scrapbook named `book`
book = sb.read_notebooks('s3://bucket/key/prefix/to/notebook/collection/')
```

The Scrapbook (book in this example) can be used to recall all scraps across the collection of notebooks:

```
book.notebook_scraps # Dict of shape `notebook` -> (`name` -> `scrap`)
book.scraps # merged dict of shape `name` -> `scrap`
```

### 3.5.1 scraps\_report

The Scrapbook collection can be used to generate a scraps\_report on all the scraps from the collection as a markdown structured output.

```
book.scraps_report()
```

This display can filter on scrap and notebook names, as well as enable or disable an overall header for the display.

```
book.scraps_report(
    scrap_names=["scrap1", "scrap2"],
    notebook_names=["result1"], # matches `/notebook/collections/result1.ipynb` pathed_
    ↪notebooks
    header=False
)
```

By default the report will only populate with visual elements. To also report on data elements set include\_data.

```
book.scraps_report(include_data=True)
```

### 3.5.2 papermill support

Finally the scrapbook has two backwards compatible features for deprecated papermill capabilities:

```
book.papermill_dataframe
book.papermill_metrics
```

## 3.6 papermill record

**scrapbook** provides a robust and flexible recording schema. This library is intended to replace [papermill](#)'s existing `record` functionality.

Documentation for `papermill record` In brief:

`pm.record(name, value)`: enabled users the ability to record values to be saved with the notebook [\[API documentation\]](#)

```
pm.record("hello", "world")
pm.record("number", 123)
pm.record("some_list", [1, 3, 5])
pm.record("some_dict", {"a": 1, "b": 2})
```

`pm.read_notebook(notebook)`: pandas could be used later to recover recorded values by reading the output notebook into a dataframe.

```
nb = pm.read_notebook('notebook.ipynb')
nb.dataframe
```

### 3.6.1 Limitations and challenges

- The `record` function didn't follow `papermill`'s pattern of linear execution of a notebook codebase. (It was awkward to describe `record` as an additional feature of `papermill` this week. It really felt like describing a second less developed library.)
- Recording / Reading required data translation to JSON for everything. This is a tedious, painful process for dataframes.
- Reading recorded values into a dataframe would result in unintuitive dataframe shapes.
- Less modularity and flexibility than other `papermill` components where custom operators can be registered.

## API REFERENCE

If you are looking for information about a specific function, class, or method, this documentation section will help you.

### 4.1 scrapbook

#### 4.1.1 scrapbook package

##### Subpackages

##### scrapbook.tests package

##### Submodules

##### scrapbook.tests.test\_api module

```
scrapbook.tests.test_api.kernel_mock()  
    Mocks the kernel to capture warnings during testing  
scrapbook.tests.test_api.test_filter_filenames(mock_read_notebook,  
                                                mock_list_notebook_files)  
scrapbook.tests.test_api.test_glue(mock_display, name, scrap, encoder, data, metadata)  
scrapbook.tests.test_api.test_glue_display_encoder_fallback(mock_registry,  
                                                            mock_display, name,  
                                                            obj, data, encoder,  
                                                            metadata, display)  
scrapbook.tests.test_api.test_glue_display_only(mock_display, name, obj, data, en-  
                                                coder, metadata, display)  
scrapbook.tests.test_api.test_glue_plus_display(mock_display, name, obj,  
                                                data_output, display_output, encoder,  
                                                data_metadata, display_metadata,  
                                                display)  
scrapbook.tests.test_api.test_glue_warning(kernel_mock)
```

**scrapbook.tests.test\_encoders module**

```
class scrapbook.tests.test_encoders.BadData
    Bases: object

class scrapbook.tests.test_encoders.BadEncoder
    Bases: object

    decode (scrap, **kwargs)

    encode (scrap, **kwargs)

    name ()

class scrapbook.tests.test_encoders.Dummy
    Bases: object

scrapbook.tests.test_encoders.registry ()

scrapbook.tests.test_encoders.test_bad_decode (registry)

scrapbook.tests.test_encoders.test_bad_encode (registry)

scrapbook.tests.test_encoders.test_decode (registry)

scrapbook.tests.test_encoders.test_determine_encoder_name (data, ex-
                                                         pected_encoder)

scrapbook.tests.test_encoders.test_determine_encoder_name_fails (data)

scrapbook.tests.test_encoders.test_encode (registry)

scrapbook.tests.test_encoders.test_json_decode (test_input, expected)

scrapbook.tests.test_encoders.test_json_decode_failures (test_input)

scrapbook.tests.test_encoders.test_json_encode (test_input, expected)

scrapbook.tests.test_encoders.test_json_encode_failures (test_input)

scrapbook.tests.test_encoders.test_missing_decode (registry)

scrapbook.tests.test_encoders.test_missing_encode (registry)

scrapbook.tests.test_encoders.test_pandas_encode_and_decode (test_input)

scrapbook.tests.test_encoders.test_registry_deregister (registry)

scrapbook.tests.test_encoders.test_registry_invalid_register (registry)

scrapbook.tests.test_encoders.test_registry_missing_deregister (registry)

scrapbook.tests.test_encoders.test_registry_register (registry)

scrapbook.tests.test_encoders.test_registry_reset (registry)

scrapbook.tests.test_encoders.test_text_decode (test_input, expected)

scrapbook.tests.test_encoders.test_text_encode (test_input, expected)

scrapbook.tests.test_encoders.test_unsupported_arrow_conversions (test_input)
```

**scrapbook.tests.test\_notebooks module**

```
class scrapbook.tests.test_notebooks.AnyDict
```

```
    Bases: object
```

```
scrapbook.tests.test_notebooks.kernel_mock()
```

```
    Mocks the kernel to capture warnings during testing
```

```
scrapbook.tests.test_notebooks.no_exec_result()
```

```
scrapbook.tests.test_notebooks.notebook_backwards_result()
```

```
scrapbook.tests.test_notebooks.notebook_result()
```

```
scrapbook.tests.test_notebooks.test_bad_ext()
```

```
scrapbook.tests.test_notebooks.test_bad_ext_for_url()
```

```
scrapbook.tests.test_notebooks.test_bad_path()
```

```
scrapbook.tests.test_notebooks.test_cell_timing(notebook_result)
```

```
scrapbook.tests.test_notebooks.test_data_scraps(notebook_result)
```

```
scrapbook.tests.test_notebooks.test_directory(notebook_result)
```

```
scrapbook.tests.test_notebooks.test_display_scraps(notebook_result)
```

```
scrapbook.tests.test_notebooks.test_empty_metadata()
```

```
scrapbook.tests.test_notebooks.test_execution_counts(notebook_result)
```

```
scrapbook.tests.test_notebooks.test_filename(notebook_result)
```

```
scrapbook.tests.test_notebooks.test_good_ext_for_url(mock_read)
```

```
scrapbook.tests.test_notebooks.test_malformed_cell_timing(no_exec_result)
```

```
scrapbook.tests.test_notebooks.test_malformed_execution_counts(no_exec_result)
```

```
scrapbook.tests.test_notebooks.test_malformed_execution_metrics(no_exec_result)
```

```
scrapbook.tests.test_notebooks.test_markdown()
```

```
scrapbook.tests.test_notebooks.test_metadata_but_empty_content()
```

```
scrapbook.tests.test_notebooks.test_missing_reggie(notebook_result)
```

```
scrapbook.tests.test_notebooks.test_missing_reggie_no_error(mock_display, notebook_result)
```

```
scrapbook.tests.test_notebooks.test_no_cells()
```

```
scrapbook.tests.test_notebooks.test_no_outputs()
```

```
scrapbook.tests.test_notebooks.test_papermill_dataframe(notebook_result)
```

```
scrapbook.tests.test_notebooks.test_papermill_metrics(notebook_result)
```

```
scrapbook.tests.test_notebooks.test_parameters(notebook_result)
```

```
scrapbook.tests.test_notebooks.test_record_scraps_collection_dataframe(notebook_backwards_result)
```

```
scrapbook.tests.test_notebooks.test_reggie_display(mock_display, notebook_result)
```

```
scrapbook.tests.test_notebooks.test_reggie_display_unattached(mock_display, notebook_result)
```

```
scrapbook.tests.test_notebooks.test_reglue_kernel_no_warning(kernel_mock,
                                                             notebook_result,
                                                             recwarn)
scrapbook.tests.test_notebooks.test_reglue_rename(mock_display, notebook_result)
scrapbook.tests.test_notebooks.test_reglue_scrap(mock_display, notebook_result)
scrapbook.tests.test_notebooks.test_reglue_scrap_unattached(mock_display, notebook_result)
scrapbook.tests.test_notebooks.test_reglue_warning(kernel_mock, notebook_result)
scrapbook.tests.test_notebooks.test_scrap_dataframe(notebook_result)
scrapbook.tests.test_notebooks.test_scraps_collection_dataframe(notebook_result)
```

## scrapbook.tests.test\_scrapbooks module

```
class scrapbook.tests.test_scrapbooks.AnyMarkdownWith(data=None, url=None,
                                                       filename=None, meta-
                                                       data=None)

    Bases: IPython.core.display.Markdown
scrapbook.tests.test_scrapbooks.kernel_mock()
    Mocks the kernel to capture warnings during testing
scrapbook.tests.test_scrapbooks.notebook_collection()
scrapbook.tests.test_scrapbooks.test_assign_from_path(notebook_collection)
scrapbook.tests.test_scrapbooks.test_notebook_scraps(notebook_collection)
scrapbook.tests.test_scrapbooks.test_papermill_dataframe(notebook_collection)
scrapbook.tests.test_scrapbooks.test_papermill_metrics(notebook_collection)
scrapbook.tests.test_scrapbooks.test_scraps(notebook_collection)
scrapbook.tests.test_scrapbooks.test_scraps_report(mock_display, notebook_collection)
scrapbook.tests.test_scrapbooks.test_scraps_report_no_headers(mock_display,
                                                                notebook_collection)
scrapbook.tests.test_scrapbooks.test_scraps_report_with_data(mock_display,
                                                                notebook_collection)
scrapbook.tests.test_scrapbooks.test_scraps_report_with_data_no_headers(mock_display,
                                                                           notebook_collection)
scrapbook.tests.test_scrapbooks.test_scraps_report_with_notebook_names(mock_display,
                                                                           notebook_collection)
scrapbook.tests.test_scrapbooks.test_scraps_report_with_scrap_and_notebook_names(mock_display,
                                                                           notebook_collection)
scrapbook.tests.test_scrapbooks.test_scraps_report_with_scrap_list_names(mock_display,
                                                                           notebook_collection)
```

```
scrapbook.tests.test_scrapbooks.test_scraps_report_with_scrap_string_name(mock_display,  
                                                                           note-  
                                                                           book_collection)
```

## scrapbook.tests.test\_scraps module

```
class scrapbook.tests.test_scraps.InvalidData  
    Bases: object
```

```
scrapbook.tests.test_scraps.test_payload_to_scrap(test_input, expected)  
scrapbook.tests.test_scraps.test_payload_to_scrap_later_version(mock_logging)  
scrapbook.tests.test_scraps.test_payload_to_scrap_later_version_mismatch(mock_logging)  
scrapbook.tests.test_scraps.test_payload_to_scrap_validation_error(test_input)  
scrapbook.tests.test_scraps.test_scrap_to_payload(test_input, expected)  
scrapbook.tests.test_scraps.test_scrap_to_payload_validation_error(test_input)
```

## scrapbook.tests.test\_utils module

```
scrapbook.tests.test_utils.test_is_kernel_true()  
scrapbook.tests.test_utils.test_not_kernel_in_ipython()
```

## Module contents

```
scrapbook.tests.get_fixture_path(*args)  
scrapbook.tests.get_notebook_dir(*args)  
scrapbook.tests.get_notebook_path(*args)
```

## Submodules

### scrapbook.api module

api.py

Provides the base API calls for scrapbook

```
scrapbook.api.glue(name, data, encoder=None, display=None)  
    Records a data value in the given notebook cell.
```

The recorded data value can be retrieved during later inspection of the output notebook.

The data type of the scraps is implied by the value type of any of the registered data encoders, but can be overwritten by setting the *encoder* argument to a particular encoder's registered name (e.g. "json").

This data is persisted by generating a display output with a special media type identifying the content storage encoder and data. These outputs are not visible in notebook rendering but still exist in the document. Scrapbook then can rehydrate the data associated with the notebook in the future by reading these cell outputs.

## Example

```
sb.glue("hello", "world") sb.glue("number", 123) sb.glue("some_list", [1, 3, 5]) sb.glue("some_dict", {"a": 1,
"b": 2}) sb.glue("non_json", df, 'arrow')
```

The scrapbook library can be used later to recover scraps (recorded values) from the output notebook

```
nb = sb.read_notebook('notebook.ipynb') nb.scraps
```

### Parameters

- **name** (*str*) – Name of the value to record.
- **data** (*any*) – The value to record. This must be an object for which an encoder’s *encodable* method returns True.
- **encoder** (*str* (*optional*)) – The name of the handler to use in persisting data in the notebook.
- **display** (*any* (*optional*)) – An indicator for persisting controlling displays for the named record.

```
scrapbook.api.read_notebook(path)
```

Returns a Notebook object loaded from the location specified at *path*.

**Parameters** *path* (*str*) – Path to a notebook *.ipynb* file.

**Returns** *notebook* – A Notebook object.

**Return type** *object*

```
scrapbook.api.read_notebooks(path, path_filter=None)
```

Returns a Scrapbook including the notebooks read from the directory specified by *path*.

### Parameters

- **path** (*str*) – Path to directory containing notebook *.ipynb* files.
- **path\_filter** (*Optional[Callable[*str*, *bool*]]*) – Func used to filter the notebook by its filename: should return True if you want to read that notebook and False otherwise

**Returns** *scrapbook* – A *Scrapbook* object.

**Return type** *object*

## scrapbook.encoders module

encoders.py

Provides the encoders for various data types to be persistable

```
class scrapbook.encoders.DataEncoderRegistry
```

Bases: *collections.abc.MutableMapping*

```
decode (scrap, **kwargs)
```

Finds the register for the given encoder and translates the scrap’s data from a string or JSON type to an object of the encoder output type.

**Parameters** *scrap* (*Scrap*) – A partially filled in scrap with data that needs decoding

```
deregister (encoder)
```

Removes a particular encoder from the registry



**Parameters** **name** (*str*) – Name of the mime subtype parsed by the encoder.

**determine\_encoder\_name** (*data*)

Determines the

**encode** (*scrap*, *\*\*kwargs*)

Finds the register for the given encoder and translates the scrap's data from an object of the encoder type to a JSON typed object.

**Parameters** **scrap** (*Scrap*) – A partially filled in scrap with data that needs encoding

**register** (*encoder*)

Registers a new name to a particular encoder

**Parameters**

- **name** (*str*) – Name of the mime subtype parsed by the encoder.
- **encoder** (*obj*) – The object which implements the required encoding functions.

**reset** ()

Resets the registry to have no encoders.

```
class scrapbook.encoders.DisplayEncoder
```

Bases: *object*

**ENCODER\_NAME** = 'display'

**decode** (*scrap*, *\*\*kwargs*)

**encodable** (*data*)

**encode** (*scrap*, *\*\*kwargs*)

**name** ()

```
class scrapbook.encoders.JsonEncoder
```

Bases: *object*

**ENCODER\_NAME** = 'json'

**decode** (*scrap*, *\*\*kwargs*)

**encodable** (*data*)

**encode** (*scrap*, *\*\*kwargs*)

**name** ()

```
class scrapbook.encoders.PandasArrowDataframeEncoder
```

Bases: *object*

**ENCODER\_NAME** = 'pandas'

**decode** (*scrap*, *\*\*kwargs*)

**encodable** (*data*)

**encode** (*scrap*, *\*\*kwargs*)

**name** ()

```
class scrapbook.encoders.TextEncoder
```

Bases: *object*

**ENCODER\_NAME** = 'text'

**decode** (*scrap*, *\*\*kwargs*)

```

encodable (data)
encode (scrap, **kwargs)
name ()

```

## scrapbook.exceptions module

```

exception scrapbook.exceptions.ScrapbookDataException (message, data_errors=None)
    Bases: scrapbook.exceptions.ScrapbookException

    Raised when a data translation exception is encountered

exception scrapbook.exceptions.ScrapbookException
    Bases: ValueError

    Raised when an exception is encountered when operating on a notebook.

exception scrapbook.exceptions.ScrapbookInvalidEncoder
    Bases: scrapbook.exceptions.ScrapbookException

    Raised when no encoder is found to transforming data

exception scrapbook.exceptions.ScrapbookMissingEncoder
    Bases: scrapbook.exceptions.ScrapbookException

    Raised when no encoder is found to transforming data

```

## scrapbook.log module

## scrapbook.models module

models.py

Provides the various model wrapper objects for scrapbook

```

class scrapbook.models.Notebook (node_or_path)
    Bases: object

    Representation of a notebook. This model is quasi-compatible with the nbformat NotebookNode object in that
    it support access to the v4 required fields from nbformat's json schema. For complete access to normal nbformat
    operations, use the node attribute of this model.

    Parameters node_or_path (nbformat.NotebookNode, str) – a notebook object, or a path to a
        notebook object

    property cell_timing
        a list of cell execution timings in cell order

        Type list

    property cells

    copy ()

    property directory
        directory name found for a notebook (nb)

        Type str

    property execution_counts
        a list of cell execution counts in cell order

```

**Type** `list`

**property filename**

filename found at the specified path

**Type** `str`

**property metadata**

**property metrics**

dataframe of cell execution counts and times

**Type** pandas dataframe

**property nbformat**

**property nbformat\_minor**

**property papermill\_dataframe**

dataframe of notebook parameters and cell scraps

**Type** pandas dataframe

**property papermill\_metrics**

**property papermill\_record\_dataframe**

dataframe of cell scraps

**Type** pandas dataframe

**property parameter\_dataframe**

dataframe of notebook parameters

**Type** pandas dataframe

**property parameters**

parameters stored in the notebook metadata

**Type** `dict`

**reglue** (*name*, *new\_name=None*, *raise\_on\_missing=True*, *unattached=False*)

Display output from a named source of the notebook.

**Parameters**

- **name** (`str`) – name of scrap object
- **new\_name** (`str`) – replacement name for scrap
- **raise\_error** (`bool`) – indicator for if the resketch should print a message or error on missing scraps
- **unattached** (`bool`) – indicator for rendering without making the display recallable as scrapbook data

**property scrap\_dataframe**

dataframe of cell scraps

**Type** pandas dataframe

**property scraps**

a dictionary of data found in the notebook

**Type** `dict`

**class** scrapbook.models.Scrapbook

Bases: `collections.abc.MutableMapping`

A collection of notebooks represented as a dictionary of notebooks

**property** metrics

a list of metrics from a collection of notebooks

Type `list`

**property** notebook\_scraps

a dictionary of the notebook scraps by key.

Type `dict`

**property** notebooks

a sorted list of associated notebooks.

Type `list`

**property** papermill\_dataframe

a list of data names from a collection of notebooks

Type `list`

**property** papermill\_metrics

**property** scraps

a dictionary of the merged notebook scraps.

Type `dict`

**scraps\_report** (*scrap\_names=None, notebook\_names=None, include\_data=False, headers=True*)

Display scraps as markdown structured outputs.

Parameters

- **scrap\_names** (*str or iterable[str] (optional)*) – the scraps to display as reported outputs
- **notebook\_names** (*str or iterable[str] (optional)*) – notebook names to use in filtering on scraps to report
- **include\_data** (*bool (default: False)*) – indicator that data-only scraps should be reported
- **header** (*bool (default: True)*) – indicator for if the scraps should render with a header

scrapbook.models.merge\_dicts (*dicts*)

## scrapbook.schemas module

schemas.py

Provides the json schema for various versions of scrapbook payloads

scrapbook.schemas.scrap\_schema (*version=1*)

## scrapbook.scrap module

scraps.py

Provides the Scrap and Scraps abstractions for housing data

**class** scrapbook.scrap.Scrap(*name, data, encoder, display*)

Bases: `tuple`

**data**

Alias for field number 1

**display**

Alias for field number 3

**encoder**

Alias for field number 2

**name**

Alias for field number 0

**class** scrapbook.scrap.Scraps(\**args*, \*\**kwargs*)

Bases: `collections.OrderedDict`

**property** data\_dict

**property** data\_scraps

**property** dataframe

dataframe of cell scraps

**Type** pandas dataframe

**property** display\_dict

**property** display\_scraps

scrapbook.scrap.payload\_to\_scrap(*payload*)

Translates data output format to a scrap

scrapbook.scrap.scrap\_to\_payload(*scrap*)

Translates scrap data to the output format

## scrapbook.utils module

utils.py

Provides the utilities for scrapbook functions and operations.

scrapbook.utils.deprecated(*version, replacement=None*)

Warns the user that something is deprecated. Removal planned in *version* release.

scrapbook.utils.is\_kernel()

Returns True if execution context is inside a kernel

scrapbook.utils.kernel\_required(*f*)

scrapbook.version module

Module contents

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`





## PYTHON MODULE INDEX

### S

- `scrapbook`, [26](#)
- `scrapbook.api`, [19](#)
- `scrapbook.encoders`, [20](#)
- `scrapbook.exceptions`, [22](#)
- `scrapbook.log`, [22](#)
- `scrapbook.models`, [22](#)
- `scrapbook.schemas`, [24](#)
- `scrapbook.scrap`s, [25](#)
- `scrapbook.tests`, [19](#)
- `scrapbook.tests.test_api`, [15](#)
- `scrapbook.tests.test_encoders`, [16](#)
- `scrapbook.tests.test_notebooks`, [17](#)
- `scrapbook.tests.test_scrapbooks`, [18](#)
- `scrapbook.tests.test_scrap`s, [19](#)
- `scrapbook.tests.test_utils`, [19](#)
- `scrapbook.utils`, [25](#)
- `scrapbook.version`, [26](#)



## A

AnyDict (class in *scrapbook.tests.test\_notebooks*), 17  
 AnyMarkdownWith (class in *scrapbook.tests.test\_scrapbooks*), 18

## B

BadData (class in *scrapbook.tests.test\_encoders*), 16  
 BadEncoder (class in *scrapbook.tests.test\_encoders*), 16

## C

cell\_timing() (*scrapbook.models.Notebook* property), 22  
 cells() (*scrapbook.models.Notebook* property), 22  
 copy() (*scrapbook.models.Notebook* method), 22

## D

data (*scrapbook.scrapbook.Scrap* attribute), 25  
 data\_dict() (*scrapbook.scrapbook.Scrap* property), 25  
 data\_scrap() (*scrapbook.scrapbook.Scrap* property), 25  
 DataEncoderRegistry (class in *scrapbook.encoders*), 20  
 dataframe() (*scrapbook.scrapbook.Scrap* property), 25  
 decode() (*scrapbook.encoders.DataEncoderRegistry* method), 20  
 decode() (*scrapbook.encoders.DisplayEncoder* method), 21  
 decode() (*scrapbook.encoders.JsonEncoder* method), 21  
 decode() (*scrapbook.encoders.PandasArrowDataframeEncoder* method), 21  
 decode() (*scrapbook.encoders.TextEncoder* method), 21  
 decode() (*scrapbook.tests.test\_encoders.BadEncoder* method), 16  
 deprecated() (in module *scrapbook.utils*), 25  
 deregister() (*scrapbook.encoders.DataEncoderRegistry* method), 20

determine\_encoder\_name() (*scrapbook.encoders.DataEncoderRegistry* method), 21  
 directory() (*scrapbook.models.Notebook* property), 22  
 display (*scrapbook.scrapbook.Scrap* attribute), 25  
 display\_dict() (*scrapbook.scrapbook.Scrap* property), 25  
 display\_scrap() (*scrapbook.scrapbook.Scrap* property), 25  
 DisplayEncoder (class in *scrapbook.encoders*), 21  
 Dummy (class in *scrapbook.tests.test\_encoders*), 16

## E

encodable() (*scrapbook.encoders.DisplayEncoder* method), 21  
 encodable() (*scrapbook.encoders.JsonEncoder* method), 21  
 encodable() (*scrapbook.encoders.PandasArrowDataframeEncoder* method), 21  
 encodable() (*scrapbook.encoders.TextEncoder* method), 21  
 encode() (*scrapbook.encoders.DataEncoderRegistry* method), 21  
 encode() (*scrapbook.encoders.DisplayEncoder* method), 21  
 encode() (*scrapbook.encoders.JsonEncoder* method), 21  
 encode() (*scrapbook.encoders.PandasArrowDataframeEncoder* method), 21  
 encode() (*scrapbook.encoders.TextEncoder* method), 22  
 encode() (*scrapbook.tests.test\_encoders.BadEncoder* method), 16  
 encoder (*scrapbook.scrapbook.Scrap* attribute), 25  
 ENCODER\_NAME (*scrapbook.encoders.DisplayEncoder* attribute), 21  
 ENCODER\_NAME (*scrapbook.encoders.JsonEncoder* attribute), 21  
 ENCODER\_NAME (*scrapbook.encoders.PandasArrowDataframeEncoder* attribute), 21

*attribute*), 21  
ENCODER\_NAME (*scrapbook.encoders.TextEncoder attribute*), 21  
execution\_counts() (*scrapbook.models.Notebook property*), 22

## F

filename() (*scrapbook.models.Notebook property*), 23

## G

get\_fixture\_path() (*in module scrapbook.tests*), 19  
get\_notebook\_dir() (*in module scrapbook.tests*), 19  
get\_notebook\_path() (*in module scrapbook.tests*), 19  
glue() (*in module scrapbook.api*), 19

## I

InvalidData (*class in scrapbook.tests.test\_scrap*), 19  
is\_kernel() (*in module scrapbook.utils*), 25

## J

JsonEncoder (*class in scrapbook.encoders*), 21

## K

kernel\_mock() (*in module scrapbook.tests.test\_api*), 15  
kernel\_mock() (*in module scrapbook.tests.test\_notebooks*), 17  
kernel\_mock() (*in module scrapbook.tests.test\_scrapbooks*), 18  
kernel\_required() (*in module scrapbook.utils*), 25

## M

merge\_dicts() (*in module scrapbook.models*), 24  
metadata() (*scrapbook.models.Notebook property*), 23  
metrics() (*scrapbook.models.Notebook property*), 23  
metrics() (*scrapbook.models.Scrapbook property*), 24  
module  
    scrapbook, 26  
    scrapbook.api, 19  
    scrapbook.encoders, 20  
    scrapbook.exceptions, 22  
    scrapbook.log, 22  
    scrapbook.models, 22  
    scrapbook.schemas, 24  
    scrapbook.scrap, 25  
    scrapbook.tests, 19  
    scrapbook.tests.test\_api, 15

scrapbook.tests.test\_encoders, 16  
scrapbook.tests.test\_notebooks, 17  
scrapbook.tests.test\_scrapbooks, 18  
scrapbook.tests.test\_scrap, 19  
scrapbook.tests.test\_utils, 19  
scrapbook.utils, 25  
scrapbook.version, 26

## N

name (*scrapbook.scrap.Scrap attribute*), 25  
name() (*scrapbook.encoders.DisplayEncoder method*), 21  
name() (*scrapbook.encoders.JsonEncoder method*), 21  
name() (*scrapbook.encoders.PandasArrowDataframeEncoder method*), 21  
name() (*scrapbook.encoders.TextEncoder method*), 22  
name() (*scrapbook.tests.test\_encoders.BadEncoder method*), 16  
nbformat() (*scrapbook.models.Notebook property*), 23  
nbformat\_minor() (*scrapbook.models.Notebook property*), 23  
no\_exec\_result() (*in module scrapbook.tests.test\_notebooks*), 17  
Notebook (*class in scrapbook.models*), 22  
notebook\_backwards\_result() (*in module scrapbook.tests.test\_notebooks*), 17  
notebook\_collection() (*in module scrapbook.tests.test\_scrapbooks*), 18  
notebook\_result() (*in module scrapbook.tests.test\_notebooks*), 17  
notebook\_scrap() (*scrapbook.models.Scrapbook property*), 24  
notebooks() (*scrapbook.models.Scrapbook property*), 24

## P

PandasArrowDataframeEncoder (*class in scrapbook.encoders*), 21  
papermill\_dataframe() (*scrapbook.models.Notebook property*), 23  
papermill\_dataframe() (*scrapbook.models.Scrapbook property*), 24  
papermill\_metrics() (*scrapbook.models.Notebook property*), 23  
papermill\_metrics() (*scrapbook.models.Scrapbook property*), 24  
papermill\_record\_dataframe() (*scrapbook.models.Notebook property*), 23  
parameter\_dataframe() (*scrapbook.models.Notebook property*), 23  
parameters() (*scrapbook.models.Notebook property*), 23

`payload_to_scrap()` (in module `scrapbook.scrap`), 25

## R

`read_notebook()` (in module `scrapbook.api`), 20

`read_notebooks()` (in module `scrapbook.api`), 20

`register()` (`scrapbook.encoders.DataEncoderRegistry` method), 21

`registry()` (in module `scrapbook.tests.test_encoders`), 16

`reglue()` (`scrapbook.models.Notebook` method), 23

`reset()` (`scrapbook.encoders.DataEncoderRegistry` method), 21

## S

`Scrap` (class in `scrapbook.scrap`), 25

`scrap_dataframe()` (`scrapbook.models.Notebook` property), 23

`scrap_schema()` (in module `scrapbook.schemas`), 24

`scrap_to_payload()` (in module `scrapbook.scrap`), 25

`scrapbook`  
module, 26

`Scrapbook` (class in `scrapbook.models`), 23

`scrapbook.api`  
module, 19

`scrapbook.encoders`  
module, 20

`scrapbook.exceptions`  
module, 22

`scrapbook.log`  
module, 22

`scrapbook.models`  
module, 22

`scrapbook.schemas`  
module, 24

`scrapbook.scrap`  
module, 25

`scrapbook.tests`  
module, 19

`scrapbook.tests.test_api`  
module, 15

`scrapbook.tests.test_encoders`  
module, 16

`scrapbook.tests.test_notebooks`  
module, 17

`scrapbook.tests.test_scrapbooks`  
module, 18

`scrapbook.tests.test_scrap`  
module, 19

`scrapbook.tests.test_utils`  
module, 19

`scrapbook.utils`  
module, 25

`scrapbook.version`  
module, 26

`ScrapbookDataException`, 22

`ScrapbookException`, 22

`ScrapbookInvalidEncoder`, 22

`ScrapbookMissingEncoder`, 22

`Scraps` (class in `scrapbook.scrap`), 25

`scraps()` (`scrapbook.models.Notebook` property), 23

`scraps()` (`scrapbook.models.Scrapbook` property), 24

`scraps_report()` (`scrapbook.models.Scrapbook` method), 24

## T

`test_assign_from_path()` (in module `scrapbook.tests.test_scrapbooks`), 18

`test_bad_decode()` (in module `scrapbook.tests.test_encoders`), 16

`test_bad_encode()` (in module `scrapbook.tests.test_encoders`), 16

`test_bad_ext()` (in module `scrapbook.tests.test_notebooks`), 17

`test_bad_ext_for_url()` (in module `scrapbook.tests.test_notebooks`), 17

`test_bad_path()` (in module `scrapbook.tests.test_notebooks`), 17

`test_cell_timing()` (in module `scrapbook.tests.test_notebooks`), 17

`test_data_scraps()` (in module `scrapbook.tests.test_notebooks`), 17

`test_decode()` (in module `scrapbook.tests.test_encoders`), 16

`test_determine_encoder_name()` (in module `scrapbook.tests.test_encoders`), 16

`test_determine_encoder_name_fails()` (in module `scrapbook.tests.test_encoders`), 16

`test_directory()` (in module `scrapbook.tests.test_notebooks`), 17

`test_display_scraps()` (in module `scrapbook.tests.test_notebooks`), 17

`test_empty_metadata()` (in module `scrapbook.tests.test_notebooks`), 17

`test_encode()` (in module `scrapbook.tests.test_encoders`), 16

`test_execution_counts()` (in module `scrapbook.tests.test_notebooks`), 17

`test_filename()` (in module `scrapbook.tests.test_notebooks`), 17

`test_filter_filenames()` (in module `scrapbook.tests.test_api`), 15

`test_glue()` (in module `scrapbook.tests.test_api`), 15

`test_glue_display_encoder_fallback()` (in module `scrapbook.tests.test_api`), 15

`test_glue_display_only()` (in module `scrapbook.tests.test_api`), 15

<code>test_glue_plus_display()</code> (in module <code>scrapbook.tests.test_api</code> ), 15	<code>test_payload_to_scrap()</code> (in module <code>scrapbook.tests.test_scrap</code> ), 19
<code>test_glue_warning()</code> (in module <code>scrapbook.tests.test_api</code> ), 15	<code>test_payload_to_scrap_later_version()</code> (in module <code>scrapbook.tests.test_scrap</code> ), 19
<code>test_good_ext_for_url()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17	<code>test_payload_to_scrap_later_version_mismatch()</code> (in module <code>scrapbook.tests.test_scrap</code> ), 19
<code>test_is_kernel_true()</code> (in module <code>scrapbook.tests.test_utils</code> ), 19	<code>test_payload_to_scrap_validation_error()</code> (in module <code>scrapbook.tests.test_scrap</code> ), 19
<code>test_json_decode()</code> (in module <code>scrapbook.tests.test_encoders</code> ), 16	<code>test_record_scrap_collection_dataframe()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17
<code>test_json_decode_failures()</code> (in module <code>scrapbook.tests.test_encoders</code> ), 16	<code>test_registry_deregister()</code> (in module <code>scrapbook.tests.test_encoders</code> ), 16
<code>test_json_encode()</code> (in module <code>scrapbook.tests.test_encoders</code> ), 16	<code>test_registry_invalid_register()</code> (in module <code>scrapbook.tests.test_encoders</code> ), 16
<code>test_json_encode_failures()</code> (in module <code>scrapbook.tests.test_encoders</code> ), 16	<code>test_registry_missing_deregister()</code> (in module <code>scrapbook.tests.test_encoders</code> ), 16
<code>test_malformed_cell_timing()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17	<code>test_registry_register()</code> (in module <code>scrapbook.tests.test_encoders</code> ), 16
<code>test_malformed_execution_counts()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17	<code>test_registry_reset()</code> (in module <code>scrapbook.tests.test_encoders</code> ), 16
<code>test_malformed_execution_metrics()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17	<code>test_reggie_display()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17
<code>test_markdown()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17	<code>test_reggie_display_unattached()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17
<code>test_metadata_but_empty_content()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17	<code>test_reggie_kernel_no_warning()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17
<code>test_missing_decode()</code> (in module <code>scrapbook.tests.test_encoders</code> ), 16	<code>test_reggie_rename()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 18
<code>test_missing_encode()</code> (in module <code>scrapbook.tests.test_encoders</code> ), 16	<code>test_reggie_scrap()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 18
<code>test_missing_reggie()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17	<code>test_reggie_scrap_unattached()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 18
<code>test_missing_reggie_no_error()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17	<code>test_reggie_warning()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 18
<code>test_no_cells()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17	<code>test_scrap_dataframe()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 18
<code>test_no_outputs()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17	<code>test_scrap_to_payload()</code> (in module <code>scrapbook.tests.test_scrap</code> ), 19
<code>test_not_kernel_in_ipython()</code> (in module <code>scrapbook.tests.test_utils</code> ), 19	<code>test_scrap_to_payload_validation_error()</code> (in module <code>scrapbook.tests.test_scrap</code> ), 19
<code>test_notebook_scrap()</code> (in module <code>scrapbook.tests.test_scrapbooks</code> ), 18	<code>test_scrap()</code> (in module <code>scrapbook.tests.test_scrapbooks</code> ), 18
<code>test_pandas_encode_and_decode()</code> (in module <code>scrapbook.tests.test_encoders</code> ), 16	<code>test_scrap_collection_dataframe()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 18
<code>test_papermill_dataframe()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17	<code>test_scrap_report()</code> (in module <code>scrapbook.tests.test_scrapbooks</code> ), 18
<code>test_papermill_dataframe()</code> (in module <code>scrapbook.tests.test_scrapbooks</code> ), 18	<code>test_scrap_report_no_headers()</code> (in module <code>scrapbook.tests.test_scrapbooks</code> ), 18
<code>test_papermill_metrics()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17	<code>test_scrap_report_with_data()</code> (in module <code>scrapbook.tests.test_scrapbooks</code> ), 18
<code>test_papermill_metrics()</code> (in module <code>scrapbook.tests.test_scrapbooks</code> ), 18	<code>test_scrap_report_with_data_no_headers()</code> (in module <code>scrapbook.tests.test_scrapbooks</code> ), 18
<code>test_parameters()</code> (in module <code>scrapbook.tests.test_notebooks</code> ), 17	<code>test_scrap_report_with_notebook_names()</code>

```
    (in module scrapbook.tests.test_scrapbooks),  
18  
test_scrap_report_with_scrap_and_notebook_names()  
    (in module scrapbook.tests.test_scrapbooks),  
18  
test_scrap_report_with_scrap_list_names()  
    (in module scrapbook.tests.test_scrapbooks),  
18  
test_scrap_report_with_scrap_string_name()  
    (in module scrapbook.tests.test_scrapbooks),  
18  
test_text_decode() (in module scrap-  
book.tests.test_encoders), 16  
test_text_encode() (in module scrap-  
book.tests.test_encoders), 16  
test_unsupported_arrow_conversions() (in  
module scrapbook.tests.test_encoders), 16  
TextEncoder (class in scrapbook.encoders), 21
```