gspread is a Python API for Google Sheets.

Features:

- Google Sheets API v4.
- Open a spreadsheet by title, key or url.
- Read, write, and format cell ranges.
- Sharing and access control.
- Batching updates.
CHAPTER 1

Installation

```
pip install gspread
```

Requirements: Python 3+.
import gspread

gc = gspread.service_account()

# Open a sheet from a spreadsheet in one go
wks = gc.open("Where is the money Lebowski?").sheet1

# Update a range of cells using the top left corner address
wks.update('A1', [[1, 2], [3, 4]])

# Or update a single cell
wks.update('B42', "it's down there somewhere, let me take another look.")

# Format the header
wks.format('A1:B1', {'textFormat': {'bold': True}})
3.1 Authentication

To access spreadsheets via Google Sheets API you need to authenticate and authorize your application.

- If you plan to access spreadsheets on behalf of a bot account use Service Account.
- If you’d like to access spreadsheets on behalf of end users (including yourself) use OAuth Client ID.

3.1.1 Enable API Access for a Project

1. Head to Google Developers Console and create a new project (or select the one you already have).
2. In the box labeled “Search for APIs and Services”, search for “Google Drive API” and enable it.
3. In the box labeled “Search for APIs and Services”, search for “Google Sheets API” and enable it.

3.1.2 For Bots: Using Service Account

A service account is a special type of Google account intended to represent a non-human user that needs to authenticate and be authorized to access data in Google APIs [sic].

Since it’s a separate account, by default it does not have access to any spreadsheet until you share it with this account. Just like any other Google account.

Here’s how to get one:

1. Enable API Access for a Project if you haven’t done it yet.
2. Go to “APIs & Services > Credentials” and choose “Create credentials > Service account key”.
3. Fill out the form
4. Click “Create” and “Done”.
5. Press “Manage service accounts” above Service Accounts.
6. Press on near recently created service account and select “Manage keys” and then click on “ADD KEY > Create new key”.

7. Select JSON key type and press “Create”.
You will automatically download a JSON file with credentials. It may look like this:

```json
{
  "type": "service_account",
  "project_id": "api-project-XXX",
  "private_key_id": "2cd ... ba4",
  "private_key": "-----BEGIN PRIVATE KEY-----
  NrDyLw ... jINQh/9
  -----END PRIVATE_KEY-----",
  "client_email": "473000000000-yoursisdifferent@developer.gserviceaccount.com",
  "client_id": "473 ... hd.apps.googleusercontent.com",
  ...
}
```

Remember the path to the downloaded credentials file. Also, in the next step you’ll need the value of client_email from this file.

6. Very important! Go to your spreadsheet and share it with a client_email from the step above. Just like you do with any other Google account. If you don’t do this, you’ll get a gsreal.exceptions.SpreadsheetNotFoundException exception when trying to access this spreadsheet from your application or a script.

7. Move the downloaded file to ~/.config/gspread/service_account.json. Windows users should put this file to %APPDATA%\gspread\service_account.json.

8. Create a new Python file with this code:

```python
import gspread

gc = gspread.service_account()

sh = gc.open("Example spreadsheet")

print(sh.sheet1.get('A1'))
```

Ta-da!

**Note:** If you want to store the credentials file somewhere else, specify the path to service_account.json in service_account():

```python
import gspread

gc = gspread.service_account(filename='path/to/the/downloaded/file.json')
```

Make sure you store the credentials file in a safe place.

For the curious, under the hood service_account() loads your credentials and authorizes gsreal. Similarly to the code that has been used for authentication prior to the gsreal version 3.6:

```python
from google.oauth2.service_account import Credentials

scopes = [
    'https://www.googleapis.com/auth/spreadsheets',
    'https://www.googleapis.com/auth/drive'
]

credentials = Credentials.from_service_account_file(
```

(continues on next page)
There is also the option to pass credentials as a dictionary:

```python
import gspread

credentials = {
    "type": "service_account",
    "project_id": "api-project-XXX",
    "private_key_id": "2cd ... ba4",
    "private_key": "-----BEGIN PRIVATE KEY-----

    NrDyLw ... jINQh/9\n-----END PRIVATE
˓→KEY-----\n",
    "client_email": "473000000000-yoursisdifferent@developer.gserviceaccount.com",
    "client_id": "473 ... hd.apps.googleusercontent.com",
    ...
}

gc = gspread.service_account_from_dict(credentials)

sh = gc.open("Example spreadsheet")

print(sh.sheet1.get('A1'))
```

**Note:** Older versions of gspread have used oauth2client. Google has deprecated it in favor of google-auth. If you’re still using oauth2client credentials, the library will convert these to google-auth for you, but you can change your code to use the new credentials to make sure nothing breaks in the future.

### 3.1.3 For End Users: Using OAuth Client ID

This is the case where your application or a script is accessing spreadsheets on behalf of an end user. When you use this scenario, your application or a script will ask the end user (or yourself if you’re running it) to grant access to the user’s data.

1. **Enable API Access for a Project** if you haven’t done it yet.
2. Go to “APIs & Services > OAuth Consent Screen.” Click the button for “Configure Consent Screen”.
   a. In the “1 OAuth consent screen” tab, give your app a name and fill the “User support email” and “Developer contact information”. Click “SAVE AND CONTINUE”.
   b. There is no need to fill in anything in the tab “2 Scopes”, just click “SAVE AND CONTINUE”.
   c. In the tab “3 Test users”, add the Google account email of the end user, typically your own Google email. Click “SAVE AND CONTINUE”.
   d. Double check the “4 Summary” presented and click “BACK TO DASHBOARD”.
3. Go to “APIs & Services > Credentials”
4. Click “+ Create credentials” at the top, then select “OAuth client ID”.
5. Select “Desktop app”, name the credentials and click “Create”. Click “Ok” in the “OAuth client created” popup.
6. Download the credentials by clicking the Download JSON button in “OAuth 2.0 Client IDs” section.

7. Move the downloaded file to ~/.config/gspread/credentials.json. Windows users should put this file to %APPDATA%\gspread\credentials.json.

Create a new Python file with this code:

```python
import gspread

gc = gspread.oauth()

sh = gc.open("Example spreadsheet")

print(sh.sheet1.get('A1'))
```

When you run this code, it launches a browser asking you for authentication. Follow the instruction on the web page. Once finished, gspread stores authorized credentials in the config directory next to credentials.json. You only need to do authorization in the browser once, following runs will reuse stored credentials.

**Note:** If you want to store the credentials file somewhere else, specify the path to credentials.json and authorized_user.json in oauth():

```python
gc = gspread.oauth(
    credentials_filename='path/to/the/credentials.json',
    authorized_user_filename='path/to/the/authorized_user.json'
)
```

Make sure you store the credentials file in a safe place.

**Attention:** Security Credentials file and authorized credentials contain sensitive data. Do not share these files with others and treat them like private keys.

If you are concerned about giving the application access to your spreadsheets and Drive, use Service Accounts.

**Note:** The user interface of Google Developers Console may be different when you’re reading this. If you find that this document is out of sync with the actual UI, please update it. Improvements to the documentation are always welcome. Click Edit on GitHub in the top right corner of the page, make it better and submit a PR.
4.1 Examples of gspread Usage

If you haven’t yet authorized your app, read Authentication first.

4.1.1 Opening a Spreadsheet

You can open a spreadsheet by its title as it appears in Google Docs:

```python
sh = gc.open('My poor gym results')
```

If you want to be specific, use a key (which can be extracted from the spreadsheet’s url):

```python
sht1 = gc.open_by_key('0BmgG6nO_6dprds1MN3d3MkdPal42WFRrdnRRUW11UFE')
```

Or, if you feel really lazy to extract that key, paste the entire spreadsheet’s url

```python
sht2 = gc.open_by_url('https://docs.google.com/spreadsheet/ccc?key=0Bm...FE&hl')
```

4.1.2 Creating a Spreadsheet

Use create() to create a new blank spreadsheet:

```python
sh = gc.create('A new spreadsheet')
```

Note: If you’re using a service account, this new spreadsheet will be visible only to this account. To be able to access newly created spreadsheet from Google Sheets with your own Google account you must share it with your email. See how to share a spreadsheet in the section below.
4.1.3 Sharing a Spreadsheet

If your email is otto@example.com you can share the newly created spreadsheet with yourself:

```python
sh.share('otto@example.com', perm_type='user', role='writer')
```

See `share()` documentation for a full list of accepted parameters.

4.1.4 Selecting a Worksheet

Select worksheet by index. Worksheet indexes start from zero:

```python
worksheet = sh.get_worksheet(0)
```

Or by title:

```python
worksheet = sh.worksheet("January")
```

Or the most common case: `Sheet1`:

```python
worksheet = sh.sheet1
```

To get a list of all worksheets:

```python
worksheet_list = sh.worksheets()
```

4.1.5 Creating a Worksheet

```python
worksheet = sh.add_worksheet(title="A worksheet", rows="100", cols="20")
```

4.1.6 Deleting a Worksheet

```python
sh.del_worksheet(worksheet)
```

4.1.7 Getting a Cell Value

Using A1 notation:

```python
val = worksheet.acell('B1').value
```

Or row and column coordinates:

```python
val = worksheet.cell(1, 2).value
```

If you want to get a cell formula:

```python
cell = worksheet.acell('B1', value_render_option='FORMULA').value
# or
cell = worksheet.cell(1, 2, value_render_option='FORMULA').value
```
4.1.8 Getting All Values From a Row or a Column

Get all values from the first row:

```python
values_list = worksheet.row_values(1)
```

Get all values from the first column:

```python
values_list = worksheet.col_values(1)
```

**Note:** So far we’ve been fetching a limited amount of data from a sheet. This works great until you need to get values from hundreds of cells or iterating over many rows or columns.

Under the hood, gspread uses Google Sheets API v4. Most of the time when you call a gspread method to fetch or update a sheet gspread produces one HTTP API call.

HTTP calls have performance costs. So if you find your app fetching values one by one in a loop or iterating over rows or columns you can improve the performance of the app by fetching data in one go.

What’s more, Sheets API v4 introduced Usage Limits (as of this writing, 500 requests per 100 seconds per project, and 100 requests per 100 seconds per user). When your application hits that limit, you get an `APIError 429 RESOURCE_EXHAUSTED`.

Here are the methods that may help you to reduce API calls:

- `get_all_values()` fetches values from all of the cells of the sheet.
- `get()` fetches all values from a range of cells.
- `batch_get()` can fetch values from multiple ranges of cells with one API call.
- `update()` lets you update a range of cells with a list of lists.
- `batch_update()` lets you update multiple ranges of cells with one API call.

4.1.9 Getting All Values From a Worksheet as a List of Lists

```python
list_of_lists = worksheet.get_all_values()
```

4.1.10 Getting All Values From a Worksheet as a List of Dictionaries

```python
list_of_dicts = worksheet.get_all_records()
```

4.1.11 Finding a Cell

Find a cell matching a string:

```python
cell = worksheet.find("Dough")
print("Found something at R%sC%s" % (cell.row, cell.col))
```

Find a cell matching a regular expression

4.1. Examples of gsread Usage
amount_re = re.compile(r'(Big|Enormous) dough')
cell = worksheet.find(amount_re)

`find` returns `None` if value is not found.

### 4.1.12 Finding All Matched Cells

Find all cells matching a string:

```python
cell_list = worksheet.findall("Rug store")
```

Find all cells matching a regexp:

```python
criteria_re = re.compile(r'(Small|Room-tiering) rug')
cell_list = worksheet.findall(criteria_re)
```

### 4.1.13 Clear A Worksheet

Clear one or multiple cell ranges at once:

```python
worksheet.batch_clear(['A1:B1', 'C2:E2', 'my_named_range'])
```

Clear the entire worksheet:

```python
worksheet.clear()
```

### 4.1.14 Cell Object

Each cell has a value and coordinates properties:

```python
value = cell.value
row_number = cell.row
column_number = cell.col
```

### 4.1.15 Updating Cells

Using A1 notation:

```python
worksheet.update('B1', 'Bingo!')
```

Or row and column coordinates:

```python
worksheet.update_cell(1, 2, 'Bingo!')
```

Update a range:

```python
worksheet.update('A1:B2', [[1, 2], [3, 4]])
```
4.1.16 Formatting

Here’s an example of basic formatting.

Set A1:B1 text format to bold:

```python
worksheet.format('A1:B1', {'textFormat': {'bold': True}})
```

Color the background of A2:B2 cell range in black, change horizontal alignment, text color and font size:

```python
worksheet.format("A2:B2", {
    "backgroundColor": {
        "red": 0.0,
        "green": 0.0,
        "blue": 0.0
    },
    "horizontalAlignment": "CENTER",
    "textFormat": {
        "foregroundColor": {
            "red": 1.0,
            "green": 1.0,
            "blue": 1.0
        },
        "fontSize": 12,
        "bold": True
    }
})
```

The second argument to `format()` is a dictionary containing the fields to update. A full specification of format options is available at `CellFormat` in Sheet API Reference.

**Tip:** `gspread-formatting` offers extensive functionality to help you when you go beyond basics.

4.1.17 Using gspread with pandas

`pandas` is a popular library for data analysis. The simplest way to get data from a sheet to a pandas DataFrame is with `get_all_records()`:

```python
import pandas as pd

dataframe = pd.DataFrame(worksheet.get_all_records())
```

Here’s a basic example for writing a dataframe to a sheet. With `update()` we put the header of a dataframe into the first row of a sheet followed by the values of a dataframe:

```python
import pandas as pd

worksheet.update([dataframe.columns.values.tolist()] + dataframe.values.tolist())
```

For advanced pandas use cases check out these libraries:

- `gspread-pandas`
- `gspread-dataframe`
4.1.18 Using gspread with NumPy

NumPy is a library for scientific computing in Python. It provides tools for working with high performance multi-dimensional arrays.

Read contents of a sheet into a NumPy array:

```python
import numpy as np
array = np.array(worksheet.get_all_values())
```

The code above assumes that your data starts from the first row of the sheet. If you have a header row in the first row, you need replace `worksheet.get_all_values()` with `worksheet.get_all_values()[1:]`.

Write a NumPy array to a sheet:

```python
import numpy as np

array = np.array([[1, 2, 3], [4, 5, 6]])

# Write the array to worksheet starting from the A2 cell
worksheet.update('A2', array.tolist())
```
Advanced Usage

5.1 Custom Authentication

Google Colaboratory

If you familiar with the Jupyter Notebook, Google Colaboratory is probably the easiest way to get started using gspread:

```python
from google.colab import auth
auth.authenticate_user()
import gspread
from oauth2client.client import GoogleCredentials
gc = gspread.authorize(GoogleCredentials.get_application_default())
```

See the full example in the External data: Local Files, Drive, Sheets, and Cloud Storage notebook.

Using Authlib

Using Authlib instead of google-auth. Similar to google.auth.transport.requests.AuthorizedSession Authlib’s AssertionSession can automatically refresh tokens:

```python
import json
from gspread import Client
from authlib.integrations.requests_client import AssertionSession

def create AssertionSession(conf_file, scopes, subject=None):
    with open(conf_file, 'r') as f:
        conf = json.load(f)
```

(continues on next page)
token_url = conf['token_uri']
issuer = conf['client_email']
key = conf['private_key']
key_id = conf.get('private_key_id')

header = {'alg': 'RS256'}
if key_id:
    header['kid'] = key_id

# Google puts scope in payload
claims = {'scope': ' '.join(scopes)}

return AssertionSession(
    grant_type=AssertionSession.JWT_BEARER_GRANT_TYPE,
    token_url=token_url,
    issuer=issuer,
    audience=token_url,
    claims=claims,
    subject=subject,
    key=key,
    header=header,
)

scopes = [
    'https://www.googleapis.com/auth/spreadsheets',
    'https://www.googleapis.com/auth/drive',
]

session = create_assertion_session('your-google-conf.json', scopes)
gc = Client(None, session)

wks = gc.open("Where is the money Lebowski?").sheet1

wks.update_acell('B2', "it's down there somewhere, let me take another look.")

# Fetch a cell range

cell_list = wks.range('A1:B7')
6.1 API Reference

6.1.1 Top level

gspread.oauth(scopes=['https://www.googleapis.com/auth/spreadsheets',
                    'https://www.googleapis.com/auth/drive'],
                   flow=<function local_server_flow>,
                   credentials_filename='/home/docs/.config/gspread/credentials.json',
                   authorized_user_filename='/home/docs/.config/gspread/authorized_user.json')

Authenticate with OAuth Client ID.

By default this function will use the local server strategy and open the authorization URL in the user’s browser:

```python
gc = gspread.oauth()
```

Another option is to run a console strategy. This way, the user is instructed to open the authorization URL in their browser. Once the authorization is complete, the user must then copy & paste the authorization code into the application:

```python
gc = gspread.oauth(flow=gspread.auth.console_flow)
```

`scopes` parameter defaults to read/write scope available in `gspread.auth.DEFAULT_SCOPES`. It’s read/write for Sheets and Drive API:

```python
default_scopes ={
    'https://www.googleapis.com/auth/spreadsheets',
    'https://www.googleapis.com/auth/drive'
}
```

You can also use `gspread.auth.READONLY_SCOPES` for read only access. Obviously any method of `gspread` that updates a spreadsheet will not work in this case:
If you’re storing your user credentials in a place other than the default, you may provide a path to that file like so:

```python
gc = gspread.oauth(
    credentials_filename='/alternative/path/credentials.json',
    authorized_user_filename='/alternative/path/authorized_user.json',
)
```

**Parameters**

- **scopes** *(list)* – The scopes used to obtain authorization.
- **flow** *(function)* – OAuth flow to use for authentication. Defaults to `local_server_flow()`.
- **credentials_filename** *(str)* – Filepath (including name) pointing to a credentials .json file. Defaults to `DEFAULT_CREDENTIALS_FILENAME`:
  - `%APPDATA%gspreadcredentials.json` on Windows
  - `~/.config/gspread/credentials.json` everywhere else
- **authorized_user_filename** *(str)* – Filepath (including name) pointing to an authorized user .json file. Defaults to `DEFAULT_AUTHORIZED_USER_FILENAME`:
  - `%APPDATA%gspreadauthorized_user.json` on Windows
  - `~/.config/gspread/authorized_user.json` everywhere else

**Return type** `gspread.Client`

```python
gspread.service_account (filename='/home/docs/.config/gspread/service_account.json',
scopes=['https://www.googleapis.com/auth/spreadsheets',
'https://www.googleapis.com/auth/drive'])
```

Authenticate using a service account.

**scopes** parameter defaults to read/write scope available in `gspread.auth.DEFAULT_SCOPES`. It’s read/write for Sheets and Drive API:

```
DEFAULT_SCOPES = [
    'https://www.googleapis.com/auth/spreadsheets',
    'https://www.googleapis.com/auth/drive'
]
```

You can also use `gspread.auth.READONLY_SCOPES` for read only access. Obviously any method of gspread that updates a spreadsheet will not work in this case.

**Parameters**

- **filename** *(str)* – The path to the service account json file.
- **scopes** *(list)* – The scopes used to obtain authorization.

**Return type** `gspread.Client`
gsheet Documentation, Release 4.0.0

**gsheet.authorize** *(credentials, client_class=<class 'gsheet.client.Client'>)*

Login to Google API using OAuth2 credentials. This is a shortcut function which instantiates `client_class`. By default `gsheet.Client` is used.

Returns `client_class` instance.

### 6.1.2 Client

```python
class gsheet.Client(auth, session=None)
```

An instance of this class communicates with Google API.

**Parameters**

- `auth` – An OAuth2 credential object. Credential objects created by `google-auth`.
- `session` – (optional) A session object capable of making HTTP requests while persisting some parameters across requests. Defaults to `google.auth.transport.requests.AuthorizedSession`.

```python
>>> c = gsheet.Client(auth=OAuthCredentialObject)
```

**copy** *(file_id, title=None, copy_permissions=False, folder_id=None)*

Copies a spreadsheet.

**Parameters**

- `file_id` *(str)* – A key of a spreadsheet to copy.
- `title` *(str)* – (optional) A title for the new spreadsheet.
- `copy_permissions` *(bool)* – (optional) If True, copy permissions from the original spreadsheet to the new spreadsheet.
- `folder_id` *(str)* – Id of the folder where we want to save the spreadsheet.

Returns a `Spreadsheet` instance.

New in version 3.1.0.

**Note:** If you’re using custom credentials without the Drive scope, you need to add `https://www.googleapis.com/auth/drive` to your OAuth scope in order to use this method.

Example:

```python
scope = [
    'https://www.googleapis.com/auth/spreadsheets',
    'https://www.googleapis.com/auth/drive'
]
```

Otherwise, you will get an **Insufficient Permission** error when you try to copy a spreadsheet.

**create** *(title, folder_id=None)*

Creates a new spreadsheet.

**Parameters**

- `title` *(str)* – A title of a new spreadsheet.
- `folder_id` *(str)* – Id of the folder where we want to save the spreadsheet.

Returns a `Spreadsheet` instance.
**del_spreadsheet** *(file_id)*  
Deletes a spreadsheet.  
*Parameters*  
**file_id** *(str)* – a spreadsheet ID (a.k.a file ID).

**import_csv** *(file_id, data)*  
Imports data into the first page of the spreadsheet.  
*Parameters*  
**data** *(str)* – A CSV string of data.

Example:
```python
# Read CSV file contents
content = open('file_to_import.csv', 'r').read()
gc.import_csv(spreadsheet.id, content)
```

**Note:** This method removes all other worksheets and then entirely replaces the contents of the first worksheet.

**insert_permission** *(file_id, value, perm_type, role, notify=True, email_message=None, with_link=False)*  
Creates a new permission for a file.  
*Parameters*  
• **file_id** *(str)* – a spreadsheet ID (aka file ID).
• **value** *(str, None)* – user or group e-mail address, domain name or None for ‘default’ type.
• **perm_type** *(str)* – (optional) The account type. Allowed values are: user, group, domain, anyone.
• **role** *(str)* – (optional) The primary role for this user. Allowed values are: owner, writer, reader.
• **notify** *(str)* – (optional) Whether to send an email to the target user/domain.
• **email_message** *(str)* – (optional) An email message to be sent if notify=True.
• **with_link** *(bool)* – (optional) Whether the link is required for this permission to be active.

Examples:
```python
# Give write permissions to otto@example.com
gc.insert_permission(
    '0BmgG6nO_6dprnRRUWl1UFE',
    'otto@example.org',
    perm_type='user',
    role='writer'
)

# Make the spreadsheet publicly readable
gc.insert_permission(
    '0BmgG6nO_6dprnRRUWl1UFE',
    None,
    with_link=True
)
```

(continues on next page)
list_permissions(file_id)
Retrieve a list of permissions for a file.

Parameters file_id (str) – a spreadsheet ID (aka file ID).

open(title)
Opens a spreadsheet.

Parameters title (str) – A title of a spreadsheet.

Returns a Spreadsheet instance.
If there’s more than one spreadsheet with same title the first one will be opened.

Raises gspread.SpreadsheetNotFound – if no spreadsheet with specified title is found.

>>> gc.open('My fancy spreadsheet')

open_by_key(key)
Opens a spreadsheet specified by key (a.k.a Spreadsheet ID).

Parameters key (str) – A key of a spreadsheet as it appears in a URL in a browser.

Returns a Spreadsheet instance.

>>> gc.open_by_key('0BmgG6nO_6dprdS1MN3d3MkdPa142WFRrdnRRUWl1UFE')

open_by_url(url)
Opens a spreadsheet specified by url.

Parameters url (str) – URL of a spreadsheet as it appears in a browser.

Returns a Spreadsheet instance.

Raises gspread.SpreadsheetNotFound – if no spreadsheet with specified url is found.

>>> gc.open_by_url('https://docs.google.com/spreadsheet/ccc?key=0Bm...FE&hl')

openall(title=None)
Opens all available spreadsheets.

Parameters title (str) – (optional) If specified can be used to filter spreadsheets by title.

Returns a list of Spreadsheet instances.

remove_permission(file_id, permission_id)
Deletes a permission from a file.

Parameters
- file_id (str) – a spreadsheet ID (aka file ID.)
- permission_id (str) – an ID for the permission.

6.1.3 Models
The models represent common spreadsheet entities: a spreadsheet, a worksheet and a cell.
Note: The classes described below should not be instantiated by the end-user. Their instances result from calling other objects’ methods.

class gspread.spreadsheet.Spreadsheet(client, properties)
The class that represents a spreadsheet.

add_worksheet (title, rows, cols, index=None)
Adds a new worksheet to a spreadsheet.

Parameters
• title (str) – A title of a new worksheet.
• rows (int) – Number of rows.
• cols (int) – Number of columns.
• index (int) – Position of the sheet.

Returns a newly created worksheets.

batch_update (body)
Lower-level method that directly calls spreadsheets.batchUpdate.

Parameters body (dict) – Request body.

Returns Response body.

Return type dict
New in version 3.0.

creationTime
Spreadsheet Creation time.

del_worksheet (worksheet)
Deletes a worksheet from a spreadsheet.

Parameters worksheet (Worksheet) – The worksheet to be deleted.

duplicate_sheet (source_sheet_id, insert_sheet_index=None, new_sheet_id=None, new_sheet_name=None)
Duplicates the contents of a sheet.

Parameters
• source_sheet_id (int) – The sheet ID to duplicate.
• insert_sheet_index (int) – (optional) The zero-based index where the new sheet should be inserted. The index of all sheets after this are incremented.
• new_sheet_id (int) – (optional) The ID of the new sheet. If not set, an ID is chosen. If set, the ID must not conflict with any existing sheet ID. If set, it must be non-negative.
• new_sheet_name (str) – (optional) The name of the new sheet. If empty, a new name is chosen for you.

Returns a newly created <gspread.models.Worksheet>.

New in version 3.1.

get_worksheet (index)
Returns a worksheet with specified index.

Parameters index (int) – An index of a worksheet. Indexes start from zero.
**Returns** an instance of `gspread.models.Worksheet`.

** Raises** `WorksheetNotFound`: if can’t find the worksheet

Example. To get third worksheet of a spreadsheet:

```python
>>> sht = client.open('My fancy spreadsheet')
>>> worksheet = sht.get_worksheet(2)
```

### `get_worksheet_by_id(id)`

Returns a worksheet with specified `worksheet id`.

**Parameters**
- **id** (*int*) – The id of a worksheet. it can be seen in the url as the value of the parameter ‘gid’.

**Returns** an instance of `gspread.models.Worksheet`.

** Raises** `WorksheetNotFound`: if can’t find the worksheet

Example. To get the worksheet 123456 of a spreadsheet:

```python
>>> sht = client.open('My fancy spreadsheet')
>>> worksheet = sht.get_worksheet_by_id(123456)
```

**id**
Spreadsheet ID.

**lastUpdateTime**
Spreadsheet Creation time.

**list_permissions()**
Lists the spreadsheet’s permissions.

**remove_permissions(value, role='any')**
Remove permissions from a user or domain.

**Parameters**
- **value** (*str*) – User or domain to remove permissions from
- **role** (*str*) – (optional) Permission to remove. Defaults to all permissions.

Example:

```python
# Remove Otto's write permission for this spreadsheet
sh.remove_permissions('otto@example.com', role='writer')

# Remove all Otto's permissions for this spreadsheet
sh.remove_permissions('otto@example.com')
```

**reorder_worksheets(worksheets_in_desired_order)**
Updates the index property of each Worksheets to reflect its index in the provided sequence of Worksheets.

**Parameters**
- **worksheets_in_desired_order** – Iterable of `Worksheet` objects in desired order.

Note: If you omit some of the Spreadsheet’s existing Worksheet objects from the provided sequence, those Worksheets will be appended to the end of the sequence in the order that they appear in the list returned by `Spreadsheet.worksheets()`.

New in version 3.4.
**share** *(value, perm_type, role, notify=True, email_message=None, with_link=False)*

Share the spreadsheet with other accounts.

**Parameters**

- **value** *(str, None)* – user or group e-mail address, domain name or None for ‘default’ type.
- **perm_type** *(str)* – The account type. Allowed values are: user, group, domain, anyone.
- **role** *(str)* – The primary role for this user. Allowed values are: owner, writer, reader.
- **notify** *(str)* – (optional) Whether to send an email to the target user/domain.
- **email_message** *(str)* – (optional) The email to be sent if notify=True
- **with_link** *(bool)* – (optional) Whether the link is required for this permission

**Example:**

```python
# Give Otto a write permission on this spreadsheet
sh.share('otto@example.com', perm_type='user', role='writer')

# Transfer ownership to Otto
sh.share('otto@example.com', perm_type='user', role='owner')
```

**sheet1**

Shortcut property for getting the first worksheet.

**title**

Spreadsheet title.

**updated**

Deprecated since version 2.0. This feature is not supported in Sheets API v4.

**url**

Spreadsheet URL.

**values_append** *(range, params, body)*

Lower-level method that directly calls spreadsheets.values.append.

**Parameters**

- **range** *(str)* – The A1 notation of a range to search for a logical table of data. Values will be appended after the last row of the table.
- **params** *(dict)* – Query parameters.
- **body** *(dict)* – Request body.

**Returns**

Response body.

**Return type** *dict*

New in version 3.0.

**values_batch_get** *(ranges, params=None)*

Lower-level method that directly calls spreadsheets.values.batchGet.

**Parameters**
values_clear (range)
Lower-level method that directly calls spreadsheets.values.clear.

Parameters range (str) – The A1 notation of the values to clear.

Returns
Response body.

Return type dict
New in version 3.0.

values_get (range, params=None)
Lower-level method that directly calls spreadsheets.values.get.

Parameters

• range (str) – The A1 notation of the values to retrieve.

• params (dict) – (optional) Query parameters.

Returns
Response body.

Return type dict
New in version 3.0.

values_update (range, params=None, body=None)
Lower-level method that directly calls spreadsheets.values.update.

Parameters

• range (str) – The A1 notation of the values to update.

• params (dict) – (optional) Query parameters.

• body (dict) – (optional) Request body.

Returns
Response body.

Return type dict

Example:

```
sh.values_update(
    'Sheet1!A2',
    params={
        'valueInputOption': 'USER_ENTERED'
    },
    body={
        'values': [[1, 2, 3]]
    }
)
```
worksheet (title)
Returns a worksheet with specified title.

Parameters title (str) – A title of a worksheet. If there’re multiple worksheets with the same title, first one will be returned.

Returns an instance of gspread.models.Worksheet.

Raises WorksheetNotFound: if can’t find the worksheet

Example. Getting worksheet named ‘Annual bonuses’

```python
>>> sht = client.open('Sample one')
>>> worksheet = sht.worksheet('Annual bonuses')
```

worksheets ()
Returns a list of all worksheets in a spreadsheet.

class gspread.worksheet.Worksheet (spreadsheet, properties)
The class that represents a single sheet in a spreadsheet (aka ”worksheet”).

acell (label, value_render_option='FORMATTED_VALUE')
Returns an instance of a gspread.models.Cell.

Parameters

* label (str) – Cell label in A1 notation Letter case is ignored.
* value_render_option (str) – (optional) Determines how values should be rendered in the output. See ValueRenderOption in the Sheets API.

Example:

```python
>>> worksheet.acell('A1')
<Cell R1C1 "I'm cell A1">
```

add_cols (cols)
Adds columns to worksheet.

Parameters cols (int) – Number of new columns to add.

add_protected_range (name, editor_users_emails=None, editor_groups_emails=None, description=None, warning_only=False, requesting_user_can_edit=False)
"Add protected range to the sheet. Only the editors can edit the protected range.

Parameters name (str) – A string with range value in A1 notation, e.g. ’A1:A5’.

Alternatively, you may specify numeric boundaries. All values index from 1 (one):

Parameters

* first_row (int) – First row number
* first_col (int) – First column number
* last_row (int) – Last row number
* last_col (int) – Last column number

For both A1 and numeric notation:

Parameters
• **editor_users_emails** (*list*) – (optional) The email addresses of users with edit access to the protected range.

• **editor_groups_emails** (*list*) – (optional) The email addresses of groups with edit access to the protected range.

• **description** (*str*) – (optional) Description for the protected ranges.

• **warning_only** (*boolean*) – (optional) When true this protected range will show a warning when editing. Defaults to False.

• **requesting_user_can_edit** (*boolean*) – (optional) True if the user who requested this protected range can edit the protected cells. Defaults to False.

`add_rows(rows)`

Adds rows to worksheet.

**Parameters**

- **rows** (*int*) – Number of new rows to add.

`append_row(values, value_input_option='RAW', insert_data_option=None, table_range=None)`

Adds a row to the worksheet and populates it with values.

Widens the worksheet if there are more values than columns.

**Parameters**

- **values** (*list*) – List of values for the new row.
- **value_input_option** (*str*) – (optional) Determines how the input data should be interpreted. See ValueInputOption in the Sheets API reference.
- **insert_data_option** (*str*) – (optional) Determines how the input data should be inserted. See InsertDataOption in the Sheets API reference.
- **table_range** (*str*) – (optional) The A1 notation of a range to search for a logical table of data. Values are appended after the last row of the table. Examples: A1 or B2:D4

`append_rows(values, value_input_option='RAW', insert_data_option=None, table_range=None)`

Adds multiple rows to the worksheet and populates them with values.

Widens the worksheet if there are more values than columns.

**Parameters**

- **values** (*list*) – List of rows each row is List of values for the new row.
- **value_input_option** (*str*) – (optional) Determines how input data should be interpreted. Possible values are RAW or USER_ENTERED. See ValueInputOption in the Sheets API.
- **insert_data_option** (*str*) – (optional) Determines how the input data should be inserted. See InsertDataOption in the Sheets API reference.
- **table_range** (*str*) – (optional) The A1 notation of a range to search for a logical table of data. Values are appended after the last row of the table. Examples: A1 or B2:D4

`batch_clear(ranges)`

Clears multiple ranges of cells with 1 API call.


Examples:
worksheet.batch_clear(['A1:B1', 'my_range'])
# Note: named ranges are defined in the scope of
# a spreadsheet, so even if 'my_range' does not belong to
# this sheet it is still updated

New in version 3.8.0.

**batch_get** *(ranges, **kwargs)*

Returns one or more ranges of values from the sheet.

**Parameters**

- **ranges** *(list)* – List of cell ranges in the A1 notation or named ranges.
- **major_dimension** *(str)* – (optional) The major dimension that results should use. Either ROWS or COLUMNS.
- **value_render_option** *(str)* – (optional) How values should be represented in the output. The default render option is FORMATTED_VALUE.
- **date_time_render_option** *(str)* – (optional) How dates, times, and durations should be represented in the output. This is ignored if value_render_option is FORMATTED_VALUE. The default dateTime render option is SERIAL_NUMBER.

New in version 3.3.

Examples:

```python
# Read values from 'A1:B2' range and 'F12' cell
worksheet.batch_get(['A1:B2', 'F12'])
```

**batch_update** *(data, **kwargs)*

Sets values in one or more cell ranges of the sheet at once.

**Parameters**

- **data** *(list)* – List of dictionaries in the form of `{range: '...', values: [[...], ...]}` where range is a target range to update in A1 notation or a named range, and values is a list of lists containing new values.
- **value_input_option** *(str)* – (optional) How the input data should be interpreted. Possible values are:
  - RAW The values the user has entered will not be parsed and will be stored as-is.
  - USER_ENTERED The values will be parsed as if the user typed them into the UI. Numbers will stay as numbers, but strings may be converted to numbers, dates, etc. following the same rules that are applied when entering text into a cell via the Google Sheets UI.

Examples:

```python
worksheet.batch_update([{  
  'range': 'A1:B1',  
  'values': [['42', '43']],  
}, {  
  'range': 'my_range',  
  'values': [['44', '45']],  
}])
# Note: named ranges are defined in the scope of
```

(continues on next page)
New in version 3.3.

**cell(row, col, value_render_option='FORMATTED_VALUE')**

Returns an instance of a `gspread.models.Cell` located at `row` and `col` column.

**Parameters**

- `row (int)` – Row number.
- `col (int)` – Column number.
- `value_render_option (str)` – (optional) Determines how values should be rendered in the output. See `ValueRenderOption` in the Sheets API.

**Example:**

```python
>>> worksheet.cell(1, 1)
<Cell R1C1 "I'm cell A1">
```

**clear()**

Clears all cells in the worksheet.

**clear_basic_filter()**

Remove the basic filter from a worksheet.

New in version 3.4.

**clear_note(cell)**

Clear a note. The note is attached to a certain cell.

**Parameters**

- `cell (str)` – A string with cell coordinates in A1 notation, e.g. ‘D7’.

Alternatively, you may specify numeric boundaries. All values index from 1 (one):

**Parameters**

- `first_row (int)` – First row number
- `first_col (int)` – First column number
- `last_row (int)` – Last row number
- `last_col (int)` – Last column number

New in version 3.7.

**col_count**

Number of columns.

**col_values(col, value_render_option='FORMATTED_VALUE')**

Returns a list of all values in column `col`.

Empty cells in this list will be rendered as `None`.

**Parameters**

- `col (int)` – Column number (one-based).
- `value_render_option (str)` – (optional) Determines how values should be rendered in the output. See `ValueRenderOption` in the Sheets API.
**copy_to** *(spreadsheet_id)*
Copies this sheet to another spreadsheet.

Parameters **spreadsheet_id** *(str)* – The ID of the spreadsheet to copy the sheet to.

Returns a dict with the response containing information about the newly created sheet.

Return type dict

**delete_columns** *(start_index, end_index=None)*
Deletes multiple columns from the worksheet at the specified index.

Parameters

- **start_index** *(int)* – Index of a first column for deletion.
- **end_index** *(int)* – Index of a last column for deletion. When end_index is not specified this method only deletes a single column at start_index.

**delete_dimension** *(dimension, start_index, end_index=None)*
Deletes multi rows from the worksheet at the specified index.

Parameters

- **dimension** *(str)* – A dimension to delete. ROWS or COLUMNS.
- **start_index** *(int)* – Index of a first row for deletion.
- **end_index** *(int)* – Index of a last row for deletion. When end_index is not specified this method only deletes a single row at start_index.

**delete_row** *(index)*
Deprecated since version 5.0.

Deletes the row from the worksheet at the specified index.

Parameters **index** *(int)* – Index of a row for deletion.

**delete_rows** *(start_index, end_index=None)*
Deletes multiple rows from the worksheet at the specified index.

Parameters

- **start_index** *(int)* – Index of a first row for deletion.
- **end_index** *(int)* – Index of a last row for deletion. When end_index is not specified this method only deletes a single row at start_index.

Example:

```python
# Delete rows 5 to 10 (inclusive)
worksheet.delete_rows(5, 10)

# Delete only the second row
worksheet.delete_rows(2)
```

**duplicate** *(insert_sheet_index=None, new_sheet_id=None, new_sheet_name=None)*
Duplicate the sheet.

Parameters

- **insert_sheet_index** *(int)* – (optional) The zero-based index where the new sheet should be inserted. The index of all sheets after this are incremented.
- **new_sheet_id** *(int)* – (optional) The ID of the new sheet. If not set, an ID is chosen. If set, the ID must not conflict with any existing sheet ID. If set, it must be non-negative.
new_sheet_name(str) – (optional) The name of the new sheet. If empty, a new name is chosen for you.

Returns a newly created <gspread.models.Worksheet>.

New in version 3.1.

export(format)

Deprecated since version 2.0.

This feature is not supported in Sheets API v4.

find(query, in_row=None, in_column=None)

Finds the first cell matching the query.

Parameters

• query (str, re.RegexObject) – A literal string to match or compiled regular expression.

• in_row (int) – (optional) One-based row number to scope the search.

• in_column (int) – (optional) One-based column number to scope the search.

Returns the first matching cell or None otherwise

findall(query, in_row=None, in_column=None)

Finds all cells matching the query.

Parameters

• query (str, re.RegexObject) – A literal string to match or compiled regular expression.

• in_row (int) – (optional) One-based row number to scope the search.

• in_column (int) – (optional) One-based column number to scope the search.

format(range_name, cell_format)

Formats a cell or a group of cells.

Parameters

• range_name (str) – Target range in the A1 notation.

• cell_format (dict) – Dictionary containing the fields to update. See CellFormat in the Sheets API for available fields.

Examples:

# Set 'A4' cell's text format to bold
worksheet.format("A4", {"textFormat": {"bold": True}})

# Color the background of 'A2:B2' cell range in black,
# change horizontal alignment, text color and font size
worksheet.format("A2:B2", {
    "backgroundColor": {
        "red": 0.0,
        "green": 0.0,
        "blue": 0.0
    },
    "horizontalAlignment": "CENTER",
    "textFormat": {
        "foregroundColor": {
            "red": 1.0,
            "green": 0.0,
            "blue": 0.0
        }
    }
})
freeze (rows=None, cols=None)
    Freeze rows and/or columns on the worksheet.

    Parameters
    • rows – Number of rows to freeze.
    • cols – Number of columns to freeze.

frozen_col_count
    Number of frozen columns.

frozen_row_count
    Number of frozen rows.

get (range_name=None, **kwargs)
    Reads values of a single range or a cell of a sheet.

    Parameters
    • range_name (str) – (optional) Cell range in the A1 notation or a named range.
    • major_dimension (str) – (optional) The major dimension that results should use. Either ROWS or COLUMNS.
    • value_render_option (str) – (optional) How values should be represented in the output. The default render option is FORMATTED_VALUE.
    • date_time_render_option (str) – (optional) How dates, times, and durations should be represented in the output. This is ignored if value_render_option is FORMATTED_VALUE. The default date_time_render_option is SERIAL_NUMBER.

Examples:

```python
# Return all values from the sheet
worksheet.get()

# Return value of 'A1' cell
worksheet.get('A1')

# Return values of 'A1:B2' range
worksheet.get('A1:B2')

# Return values of 'my_range' named range
worksheet.get('my_range')
```

New in version 3.3.

get_all_records (empty2zero=False, head=1, default_blank='', allow_underscores_in_numeric_literals=False, numericise_ignore=None, value_render_option=None)
Returns a list of dictionaries, all of them having the contents of the spreadsheet with the head row as keys and each of these dictionaries holding the contents of subsequent rows of cells as values.

Cell values are numericised (strings that can be read as ints or floats are converted), unless specified in numericise_ignore

Parameters

- `empty2zero (bool)` – (optional) Determines whether empty cells are converted to zeros.
- `head (int)` – (optional) Determines which row to use as keys, starting from 1 following the numeration of the spreadsheet.
- `default_blank (str)` – (optional) Determines which value to use for blank cells, defaults to empty string.
- `allow_underscores_in_numeric_literals (bool)` – (optional) Allow underscores in numeric literals, as introduced in PEP 515
- `numericise_ignore (list)` – (optional) List of ints of indices of the row (starting at 1) to ignore numericising, special use of ['all'] to ignore numericising on all columns.
- `value_render_option (str)` – (optional) Determines how values should be rendered in the the output. See ValueRenderOption in the Sheets API.

`get_all_values (**kwargs)`

Returns a list of lists containing all cells’ values as strings.

This is an alias to `get_values()`

Note: This is a legacy method. Use `get_values()` instead.

Examples:

```python
# Return all values from the sheet
worksheet.get_all_values()

# Is equivalent to
worksheet.get_values()
```

`get_note (cell)`

Get the content of the note located at `cell`, or the empty string if the cell does not have a note.

Parameters `cell (str)` – A string with cell coordinates in A1 notation, e.g. ‘D7’.

`get_values (range_name=None, **kwargs)`

Returns a list of lists containing all values from specified range.

By default values are returned as strings. See `value_render_option` to change the default format.

Parameters

- `range_name (str)` – (optional) Cell range in the A1 notation or a named range. If not specified the method returns values from all non empty cells.
- `major_dimension (str)` – (optional) The major dimension of the values. Either ROWS or COLUMNS. Defaults to ROWS
- `value_render_option (str)` – (optional) Determines how values should be rendered in the the output. See ValueRenderOption in the Sheets API.

Possible values are:
**Formatting Options**

- **FORMATTED_VALUE** (default) Values will be calculated and formatted according to the cell’s formatting. Formatting is based on the spreadsheet’s locale, not the requesting user’s locale.

- **UNFORMATTED_VALUE** Values will be calculated, but not formatted in the reply. For example, if A1 is 1.23 and A2 is =A1 and formatted as currency, then A2 would return the number 1.23.

- **FORMULA** Values will not be calculated. The reply will include the formulas. For example, if A1 is 1.23 and A2 is =A1 and formatted as currency, then A2 would return “=A1”.

**Parameters**

- **date_time_render_option** *(str)* – (optional) How dates, times, and durations should be represented in the output. This is ignored if `value_render_option` is **FORMATTED_VALUE**. The default `date_time_render_option` is **SERIAL_NUMBER**.

**Note:** Empty trailing rows and columns will not be included.

**Examples:**

```python
# Return all values from the sheet
worksheet.get_values()

# Return all values from columns "A" and "B"
worksheet.get_values('A:B')

# Return values from range "A2:C10"
worksheet.get_values('A2:C10')

# Return values from named range "my_range"
worksheet.get_values('my_range')

# Return unformatted values (e.g. numbers as numbers)
worksheet.get_values('A2:B4', value_render_option='UNFORMATTED_VALUE')

# Return cell values without calculating formulas
worksheet.get_values('A2:B4', value_render_option='FORMULA')
```

**id**

Worksheet ID.

**index**

Worksheet index.

**insert_cols** *(values, col=1, value_input_option='RAW')*

Adds multiple new cols to the worksheet at specified index and populates them with values.

**Parameters**

- **values** *(list)* – List of col lists. a list of lists, with the lists each containing one col’s values. Increases the number of rows if there are more values than columns.

- **col** *(int)* – Start col to update (one-based). Defaults to 1 (one).

- **value_input_option** *(str)* – (optional) Determines how input data should be interpreted. Possible values are **RAW** or **USER_ENTERED**. See `ValueInputOption` in the Sheets API.
**insert_note** *(cell, content)*

Insert a note. The note is attached to a certain cell.

**Parameters**

- **cell** *(str)* – A string with cell coordinates in A1 notation, e.g. ‘D7’.
- **content** *(str)* – The text note to insert.

Alternatively, you may specify numeric boundaries. All values index from 1 (one):

**Parameters**

- **first_row** *(int)* – First row number
- **first_col** *(int)* – First column number
- **last_row** *(int)* – Last row number
- **last_col** *(int)* – Last column number

New in version 3.7.

**insert_row** *(values, index=1, value_input_option=’RAW’)*

Adds a row to the worksheet at the specified index and populates it with values.

Widens the worksheet if there are more values than columns.

**Parameters**

- **values** *(list)* – List of values for the new row.
- **index** *(int)* – (optional) Offset for the newly inserted row.
- **value_input_option** *(str)* – (optional) Determines how input data should be interpreted. Possible values are RAW or USER_ENTERED. See ValueInputOption in the Sheets API.

**insert_rows** *(values, row=1, value_input_option=’RAW’)*

Adds multiple rows to the worksheet at the specified index and populates them with values.

**Parameters**

- **values** *(list)* – List of row lists. a list of lists, with the lists each containing one row’s values. Widens the worksheet if there are more values than columns.
- **row** *(int)* – Start row to update (one-based). Defaults to 1 (one).
- **value_input_option** *(str)* – (optional) Determines how input data should be interpreted. Possible values are RAW or USER_ENTERED. See ValueInputOption in the Sheets API.

**merge_cells** *(name, merge_type=’MERGE_ALL’)*

Merge cells. There are 3 merge types: MERGE_ALL, MERGE_COLUMNS, and MERGE_ROWS.

**Parameters**

- **name** *(str)* – Range name in A1 notation, e.g. ‘A1:A5’.
- **merge_type** *(str)* – (optional) one of MERGE_ALL, MERGE_COLUMNS, or MERGE_ROWS. Defaults to MERGE_ROWS. See MergeType in the Sheets API reference.

Alternatively, you may specify numeric boundaries. All values index from 1 (one):

**Parameters**

- **first_row** *(int)* – First row number
• **first_col** (*int*) – First column number
• **last_row** (*int*) – Last row number
• **last_col** (*int*) – Last column number

**Returns** the response body from the request

**Return type** dict

### range(*name*)

Returns a list of `Cell` objects from a specified range.

**Parameters**

- **name** (*str*) – A string with range value in A1 notation (e.g. ‘A1:A5’) or the named range to fetch.

Alternatively, you may specify numeric boundaries. All values index from 1 (one):

**Parameters**

- **first_row** (*int*) – First row number
- **first_col** (*int*) – First column number
- **last_row** (*int*) – Last row number
- **last_col** (*int*) – Last column number

**Example:**

```python
>>> # Using A1 notation
>>> worksheet.range('A1:B7')
[<Cell R1C1 "42">, ...]

>>> # Same with numeric boundaries
>>> worksheet.range(1, 1, 7, 2)
[<Cell R1C1 "42">, ...]

>>> # Named ranges work as well
>>> worksheet.range('NamedRange')
[<Cell R1C1 "42">, ...]
```

### resize(*rows=None, cols=None*)

Resizes the worksheet. Specify one of **rows** or **cols**.

**Parameters**

- **rows** (*int*) – (optional) New number of rows.
- **cols** (*int*) – (optional) New number columns.

### row_count

Number of rows.

### row_values(*row, **kwargs*)

Returns a list of all values in a **row**.

Empty cells in this list will be rendered as `None`.

**Parameters**

- **row** (*int*) – Row number (one-based).
- **value_render_option** (*str*) – (optional) Determines how values should be rendered in the output. See `ValueRenderOption` in the Sheets API.
set_basic_filter\((name=None)\)
Add a basic filter to the worksheet. If a range or boundaries are passed, the filter will be limited to the given range.

Parameters
- name (str) – A string with range value in A1 notation, e.g. A1:A5.
Alternatively, you may specify numeric boundaries. All values index from 1 (one):

Parameters
- first_row (int) – First row number
- first_col (int) – First column number
- last_row (int) – Last row number
- last_col (int) – Last column number
New in version 3.4.

sort (*specs, **kwargs)
Sorts worksheet using given sort orders.

Parameters
- specs (list) – The sort order per column. Each sort order represented by a tuple where the first element is a column index and the second element is the order itself: ‘asc’ or ‘des’.
- range (str) – The range to sort in A1 notation. By default sorts the whole sheet excluding frozen rows.

Example:
```
# Sort sheet A -> Z by column 'B'
wks.sort((2, 'asc'))

# Sort range A2:G8 basing on column 'G' A -> Z
# and column 'B' Z -> A
wks.sort((7, 'asc'), (2, 'des'), range='A2:G8')
```
New in version 3.4.

title
Worksheet title.

update\(\)\(range\_name, values=None, **kwargs)\)
Sets values in a cell range of the sheet.

Parameters
- range_name (str) – The A1 notation of the values to update.
- values (list) – The data to be written.
- raw (bool) – The values will not be parsed by Sheets API and will be stored as-is. For example, formulas will be rendered as plain strings. Defaults to True. This is a shortcut for the value_input_option parameter.
- major_dimension (str) – (optional) The major dimension of the values. Either ROWS or COLUMNS.
- value_input_option (str) – (optional) How the input data should be interpreted. Possible values are:
- RAW The values the user has entered will not be parsed and will be stored as-is.
The values will be parsed as if the user typed them into the UI. Numbers will stay as numbers, but strings may be converted to numbers, dates, etc. following the same rules that are applied when entering text into a cell via the Google Sheets UI.

Examples:

```python
# Sets 'Hello world' in 'A2' cell
worksheet.update('A2', 'Hello world')

# Updates cells A1, B1, C1 with values 42, 43, 44 respectively
worksheet.update([42, 43, 44])

# Updates A2 and A3 with values 42 and 43
# Note that update range can be bigger than values array
worksheet.update('A2:B4', [[42], [43]])

# Add a formula
worksheet.update('A5', '=SUM(A1:A4)', raw=False)

# Update 'my_range' named range with values 42 and 43
worksheet.update('my_range', [[42], [43]])

# Note: named ranges are defined in the scope of
# a spreadsheet, so even if 'my_range' does not belong to
# this sheet it is still updated
```

New in version 3.3.

`update_acell(label, value)`

Updates the value of a cell.

**Parameters**

- **label** (`str`) – Cell label in A1 notation.
- **value** – New value.

**Example:**

```python
worksheet.update_acell('A1', '42')
```

`update_cell(row, col, value)`

Updates the value of a cell.

**Parameters**

- **row** (`int`) – Row number.
- **col** (`int`) – Column number.
- **value** – New value.

**Example:**

```python
worksheet.update_cell(1, 1, '42')
```

`update_cells(cell_list, value_input_option='RAW')`

Updates many cells at once.

**Parameters**

- **cell_list** (`list`) – List of Cell objects to update.
• **value_input_option** *(str)* – (optional) How the input data should be interpreted.

   Possible values are:

   **RAW** The values the user has entered will not be parsed and will be stored as-is.

   **USER_ENTERED** The values will be parsed as if the user typed them into the UI. Numbers will stay as numbers, but strings may be converted to numbers, dates, etc. following the same rules that are applied when entering text into a cell via the Google Sheets UI.

   See **ValueInputOption** in the Sheets API.

   **Example:**

   ```python
   # Select a range
   cell_list = worksheet.range('A1:C7')
   for cell in cell_list:
       cell.value = 'O_o'
   # Update in batch
   worksheet.update_cells(cell_list)
   ```

   **update_index**(index)

   Updates the index property for the worksheet.

   See the Sheets API documentation for information on how updating the index property affects the order of worksheets in a spreadsheet.

   To reorder all worksheets in a spreadsheet, see **Spreadsheet.reorder_worksheets**.

   New in version 3.4.

   **update_note**(cell, content)

   Update the content of the note located at cell.

   **Parameters**

   • **cell** *(str)* – A string with cell coordinates in A1 notation, e.g. ‘D7’.

   • **note** *(str)* – The text note to insert.

   New in version 3.7.

   **update_title**(title)

   Renames the worksheet.

   **Parameters**

   • **title** *(str)* – A new title.

   **updated**

   Deprecated since version 2.0.

   This feature is not supported in Sheets API v4.

   **url**

   Worksheet URL.

   **class** **gspread.cell.Cell**(row, col, value=“”)

   An instance of this class represents a single cell in a worksheet.

   **address**

   Cell address in A1 notation.

   **col**

   Column number of the cell.
input_value
Depreciated since version 2.0.
This feature is not supported in Sheets API v4.

row
Row number of the cell.

value = None
Value of the cell.

6.1.4 Utils

gspread.utils
This module contains utility functions.

gspread.utils.rowcol_to_a1(row, col)
Translates a row and column cell address to A1 notation.

Parameters
• row (int, str) – The row of the cell to be converted. Rows start at index 1.
• col – The column of the cell to be converted. Columns start at index 1.

Returns a string containing the cell’s coordinates in A1 notation.

Example:
>>> rowcol_to_a1(1, 1)
A1

gspread.utils.a1_to_rowcol(label)
Translates a cell’s address in A1 notation to a tuple of integers.

Parameters label(str) – A cell label in A1 notation, e.g. ‘B1’. Letter case is ignored.

Returns a tuple containing row and column numbers. Both indexed from 1 (one).

Return type tuple

Example:
>>> a1_to_rowcol('A1')
(1, 1)

gspread.utils.al_range_to_grid_range(name, sheet_id=None)
Converts a range defined in A1 notation to a dict representing a GridRange.

All indexes are zero-based. Indexes are half open, e.g the start index is inclusive and the end index is exclusive: [startIndex, endIndex).

Missing indexes indicate the range is unbounded on that side.

Examples:
>>> al_range_to_grid_range('A1:A1')
{'startRowIndex': 0, 'endRowIndex': 1, 'startColumnIndex': 0, 'endColumnIndex': 1}
>>> al_range_to_grid_range('A3:B4')
{'startRowIndex': 2, 'endRowIndex': 4, 'startColumnIndex': 0, 'endColumnIndex': 2}

>>> al_range_to_grid_range('A:B')
{'startColumnIndex': 0, 'endColumnIndex': 2}

>>> al_range_to_grid_range('A5:B')
{'startRowIndex': 4, 'startColumnIndex': 0, 'endColumnIndex': 2}

>>> al_range_to_grid_range('A1')
{'startRowIndex': 0, 'endRowIndex': 1, 'startColumnIndex': 0, 'endColumnIndex': 1}

>>> al_range_to_grid_range('A')
{'startColumnIndex': 0, 'endColumnIndex': 1}

>>> al_range_to_grid_range('1')
{'startRowIndex': 0, 'endRowIndex': 1}

>>> al_range_to_grid_range('A1', sheet_id=0)
{'sheetId': 0, 'startRowIndex': 0, 'endRowIndex': 1, 'startColumnIndex': 0, 'endColumnIndex': 1}

gspread.utils.cast_to_a1_notation(method)
Decorator function casts wrapped arguments to A1 notation in range method calls.

gspread.utils.absolute_range_name(sheet_name, range_name=None)
Return an absolutized path of a range.

>>> absolute_range_name("Sheet1", "A1:B1")
"'Sheet1'!A1:B1"

>>> absolute_range_name("Sheet1", "A1")
"'Sheet1'!A1"

>>> absolute_range_name("Sheet1")
"'Sheet1'"

>>> absolute_range_name("Sheet'1")
"'Sheet''1'"

>>> absolute_range_name("'sheet1'")
"'sheet1'"

>>> absolute_range_name("''sheet1''", "A1:B2")
"'''sheet1''''!A1:B2"

gspread.utils.is_scalar(x)
Return True if the value is scalar.
A scalar is not a sequence but can be a string.

>>> is_scalar([])
False
>>> is_scalar([1, 2])
False

>>> is_scalar(42)
True

>>> is_scalar('nice string')
True

>>> is_scalar({})
True

>>> is_scalar(set())
True

gspread.utils.filter_dict_values(D)
Return a shallow copy of D with all None values excluded.

>>> filter_dict_values({'a': 1, 'b': 2, 'c': None})
{'a': 1, 'b': 2}

>>> filter_dict_values({'a': 1, 'b': 2, 'c': 0})
{'a': 1, 'b': 2, 'c': 0}

>>> filter_dict_values({})
{}

>>> filter_dict_values({'imnone': None})
{}


gspread.utils.accepted_kwargs(**default_kwargs)

>>> @accepted_kwargs(d='d', e=None)
... def foo(a, b, c='c', **kwargs):
...     return {'
...     'a': a,
...     'b': b,
...     'c': c,
...     'd': kwargs['d'],
...     'e': kwargs['e'],
...     }
...

>>> foo('a', 'b')
{'a': 'a', 'b': 'b', 'c': 'c', 'd': 'd', 'e': None}

>>> foo('a', 'b', 'NEW C')
{'a': 'a', 'b': 'b', 'c': 'NEW C', 'd': 'd', 'e': None}

>>> foo('a', 'b', e='Not None')
{'a': 'a', 'b': 'b', 'c': 'c', 'd': 'd', 'e': 'Not None'}
>>> foo('a', 'b', d='NEW D')
{'a': 'a', 'b': 'b', 'c': 'c', 'd': 'NEW D', 'e': None}

>>> foo('a', 'b', a_typo='IS DETECTED')
Traceback (most recent call last):
  ...TypeError: foo got unexpected keyword arguments: ['a_typo']

>>> foo('a', 'b', d='NEW D', c='THIS DOES NOT WORK BECAUSE OF d')
Traceback (most recent call last):
  ...TypeError: foo got unexpected keyword arguments: ['c']

6.1.5 Auth

gspread.auth

Simple authentication with OAuth.

gspread.auth.local_server_flow(scopes, port=0, filename='/home/docs/.config/gspread/credentials.json')
Run an OAuth flow using a local server strategy.

  Creates an OAuth flow and runs google_auth_oauthlib.flow.InstalledAppFlow.run_local_server. This will start
  a local web server and open the authorization URL in the user’s browser.

  Pass this function to flow parameter of oauth() to run a local server flow.

gspread.auth.console_flow(scopes, filename='/home/docs/.config/gspread/credentials.json')
Run an OAuth flow using a console strategy.

  Creates an OAuth flow and runs google_auth_oauthlib.flow.InstalledAppFlow.run_console.

  Pass this function to flow parameter of oauth() to run a console strategy.

6.1.6 Exceptions

exception gspread.exceptions.GSpreadException
  A base class for gspread’s exceptions.

exception gspread.exceptions.APIError(response)
CHAPTER 7

How to Contribute

Please make sure to take a moment and read the Code of Conduct.

7.1 Ask Questions

The best way to get an answer to a question is to ask on Stack Overflow with a gsread tag.

7.2 Report Issues

Please report bugs and suggest features via the GitHub Issues.
Before opening an issue, search the tracker for possible duplicates. If you find a duplicate, please add a comment saying that you encountered the problem as well.

7.3 Contribute code

Please make sure to read the Contributing Guide before making a pull request.
CHAPTER 8

Indices and tables

- genindex
- modindex
- search
Python Module Index

g

gspread, 19

gspread.auth, 45

gspread.utils, 42
Index

A
a1_range_to_grid_range () (in module gspread.utils), 42
a1_to_rowcol () (in module gspread.utils), 42
absolute_range_name () (in module gspread.utils), 43
accepted_kwargs () (in module gspread.utils), 44
acell () (gspread.worksheet.Worksheet method), 28
add_cols () (gspread.worksheet.Worksheet method), 28
add_protected_range () (gspread.worksheet.Worksheet method), 28
add_rows () (gspread.worksheet.Worksheet method), 29
address (gspread.cell.Cell attribute), 41
APIError, 45
append_row () (gspread.worksheet.Worksheet method), 29
append_rows () (gspread.worksheet.Worksheet method), 29
authorize () (in module gspread), 20

B
batch_clear () (gspread.worksheet.Worksheet method), 29
batch_get () (gspread.worksheet.Worksheet method), 30
batch_update () (gspread.spreadsheet.Spreadsheet method), 24
batch_update () (gspread.worksheet.Worksheet method), 30

C
cast_to_a1_notation () (in module gspread.utils), 43
Cell (class in gspread.cell), 41
cell () (gspread.worksheet.Worksheet method), 31
clear () (gspread.worksheet.Worksheet method), 31
clear_basic_filter () (gspread.worksheet.Worksheet method), 31
clear_note () (gspread.worksheet.Worksheet method), 31
Client (class in gspread), 21
col (gspread.cell.Cell attribute), 41
col_count (gspread.worksheet.Worksheet attribute), 31
col_values () (gspread.worksheet.Worksheet attribute), 31
console_flow () (in module gspread.auth), 45
copy () (gspread.Client method), 21
copy_to () (gspread.worksheet.Worksheet method), 31
create () (gspread.Client method), 21
creationTime (gspread.spreadsheet.Spreadsheet attribute), 24

del_spreadsheet () (gspread.Client method), 21
del_worksheet () (gspread.spreadsheet.Spreadsheet method), 24
delete_columns () (gspread.worksheet.Worksheet method), 32
delete_dimension () (gspread.worksheet.Worksheet method), 32
delete_row () (gspread.worksheet.Worksheet method), 32
delete_rows () (gspread.worksheet.Worksheet method), 32
duplicate () (gspread.worksheet.Worksheet method), 32
duplicate_sheet () (gspread.spreadsheet.Spreadsheet method), 24
export () (gspread.worksheet.Worksheet method), 33
update_note() (gspread.worksheet.Worksheet method), 41
update_title() (gspread.worksheet.Worksheet method), 41
updated (gspread.spreadsheet.Spreadsheet attribute), 26
updated (gspread.worksheet.Worksheet attribute), 41
url (gspread.spreadsheet.Spreadsheet attribute), 26
url (gspread.worksheet.Worksheet attribute), 41

V
value (gspread.cell.Cell attribute), 42
values_append() (gspread.spreadsheet.Spreadsheet method), 26
values_batch_get() (gspread.spreadsheet.Spreadsheet method), 26
values_clear() (gspread.spreadsheet.Spreadsheet method), 27
values_get() (gspread.spreadsheet.Spreadsheet method), 27
values_update() (gspread.spreadsheet.Spreadsheet method), 27

W
Worksheet (class in gspread.worksheet), 28
worksheet() (gspread.spreadsheet.Spreadsheet method), 28
worksheets() (gspread.spreadsheet.Spreadsheet method), 28