# 14
# WORKING WITH GOOGLE SHEETS

Google Sheets, the free, web-based spreadsheet application available to anyone with a Google account or Gmail address, has become a useful, feature-rich competitor to Excel. Google Sheets has its own API, but this API can be confusing to learn and use. This chapter covers the EZSheets third-party module, documented at *https://ezsheets.readthedocs.io/*. While not as full featured as the official Google Sheets API, EZSheets makes common spreadsheet tasks easy to perform.

## INSTALLING AND SETTING UP EZSHEETS

You can install EZSheets by opening a new terminal window and running `pip install --user ezsheets`. As part of this installation, EZSheets will also install the `google-api-python-client`, `google-auth-httplib2`, and `google-auth-oauthlib` modules. These modules allow your program to log in to Google's servers and make API requests. EZSheets handles the interaction with these modules, so you don't need to concern yourself with how they work.

### *Obtaining Credentials and Token Files*

Before you can use EZSheets, you need to enable the Google Sheets and Google Drive APIs for your Google account. Visit the following web pages and click the **Enable API** buttons at the top of each:

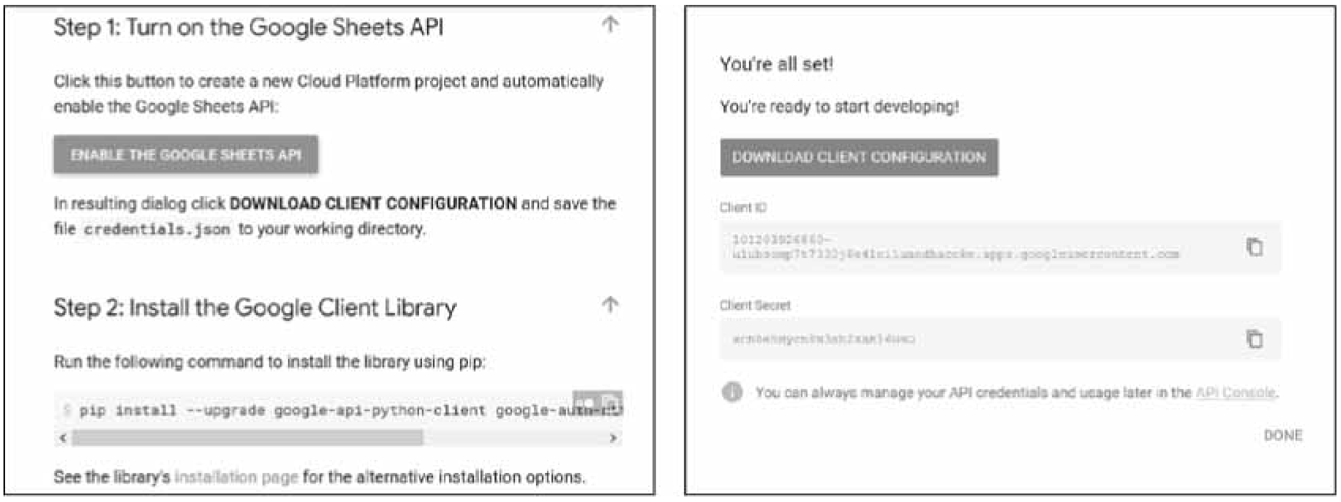
- *https://console.developers.google.com/apis/library/sheets.googleapis.com/*

- *https://console.developers.google.com/apis/library/drive.googleapis.com/*

You'll also need to obtain three files, which you should save in the same folder as your *.py* Python script that uses EZSheets:

- A credentials file named *credentials-sheets.json*
- A token for Google Sheets named *token-sheets.pickle*
- A token for Google Drive named *token-drive.pickle*

The credentials file will generate the token files. The easiest way to obtain a credentials file is to go to the Google Sheets Python Quickstart page at *https://developers.google.com/sheets/api/quickstart/python/* and click the blue **Enable the Google Sheets API** button, as shown in Figure 14-1. You'll need to log in to your Google account to view this page.



Step 1: Turn on the Google Sheets API

Click this button to create a new Cloud Platform project and automatically enable the Google Sheets API:

ENABLE THE GOOGLE SHEETS API

In resulting dialog click **DOWNLOAD CLIENT CONFIGURATION** and save the file `credentials.json` to your working directory.

Step 2: Install the Google Client Library

Run the following command to install the library using pip:

```
$ pip install --upgrade google-api-python-client google-auth-
```

See the library's installation page for the alternative installation options.

You're all set!

You're ready to start developing!

DOWNLOAD CLIENT CONFIGURATION

Client ID

[illegible]

Client Secret

[illegible]

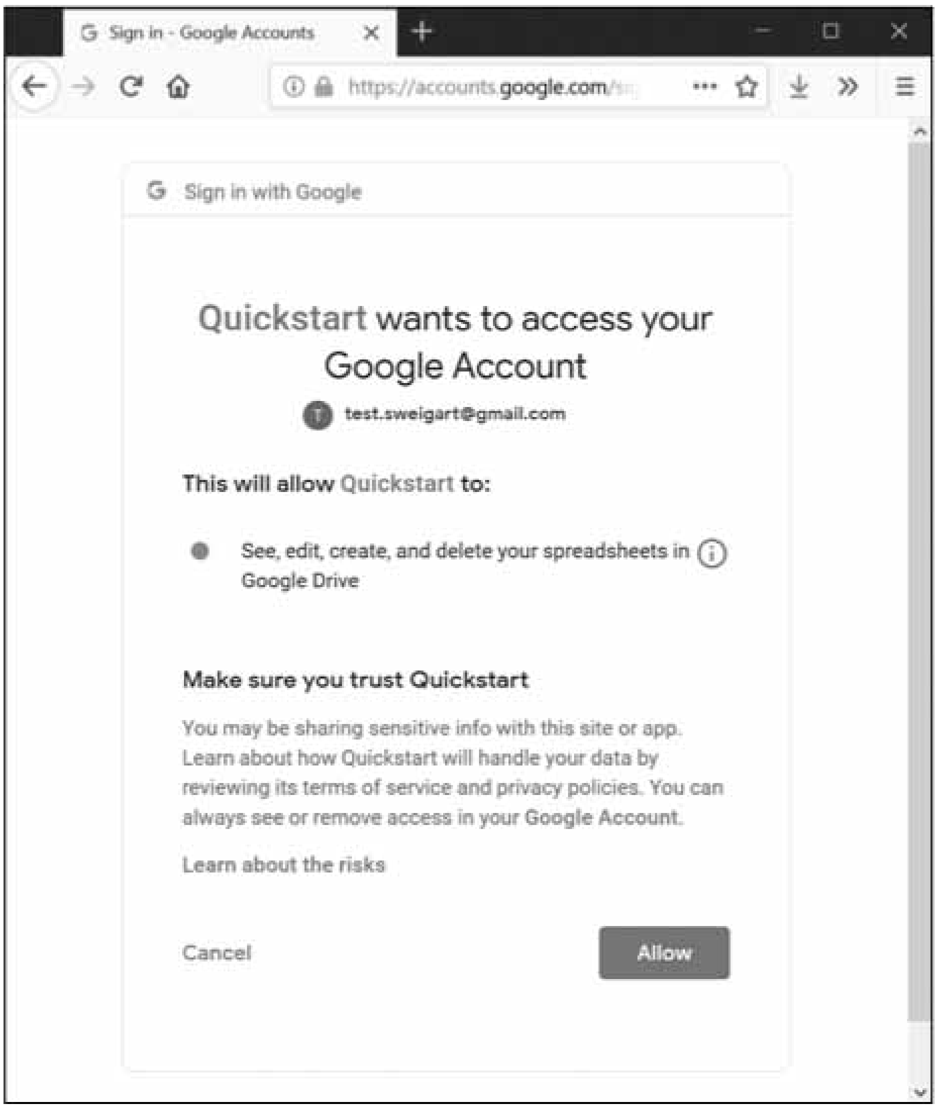
You can always manage your API credentials and usage later in the API Console.

DONE

*Figure 14-1: Obtaining a* credentials.json *file.*

Clicking this button will bring up a window with a **Download Client Configuration** link that lets you download a *credentials.json* file. Rename this file to *credentials-sheets.json* and place it in the same folder as your Python scripts.

Once you have a *credentials-sheets.json* file, run the `import ezsheets` module. The first time you import the EZSheets module, it will open a new browser window for you to log in to your Google account. Click **Allow**, as shown in Figure 14-2.



Quickstart wants to access your Google Account

test.sweigart@gmail.com

This will allow Quickstart to:

- See, edit, create, and delete your spreadsheets in Google Drive

Make sure you trust Quickstart

You may be sharing sensitive info with this site or app. Learn about how Quickstart will handle your data by reviewing its terms of service and privacy policies. You can always see or remove access in your Google Account.

Learn about the risks

*Figure 14-2: Allowing Quickstart to access your Google account*

The message about Quickstart comes from the fact that you downloaded the credentials file from the Google Sheets Python Quickstart page. Note that this window will open *twice*: first for Google Sheets access and second for Google Drive access. EZSheets uses Google Drive access to upload, download, and delete spreadsheets.

After you log in, the browser window will prompt you to close it, and the *token-sheets.pickle* and *token-drive.pickle* files will appear in the same folder as *credentials-sheets.json*. You only need to go through this process the first time you run `import ezsheets`.

If you encounter an error after clicking Allow and the page seems to hang, make sure you have first enabled the Google Sheets and Drive APIs from the links at the start of this section. It may take a few minutes for Google's servers to register this change, so you may have to wait before you can use EZSheets.

Don't share the credential or token files with anyone—treat them like passwords.

## *Revoking the Credentials File*

If you accidentally share the credential or token files with someone, they won't be able to change your Google account password, but they will have access to your spreadsheets.

You can revoke these files by going to the Google Cloud Platform developer's console page at *https://console.developers.google.com/*. You'll need to log in to your Google account to view this page. Click the **Credentials** link on the sidebar. Then click the trash can icon next to the credentials file you've accidentally shared, as shown in Figure 14-3.

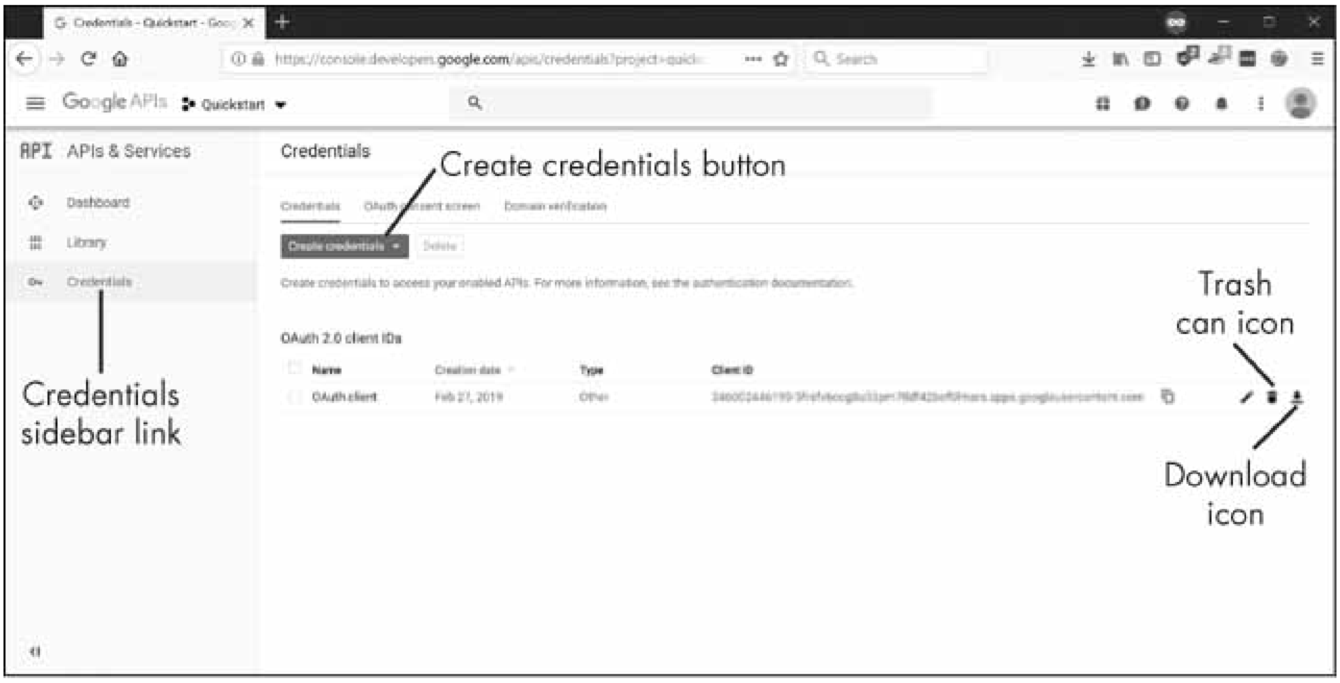
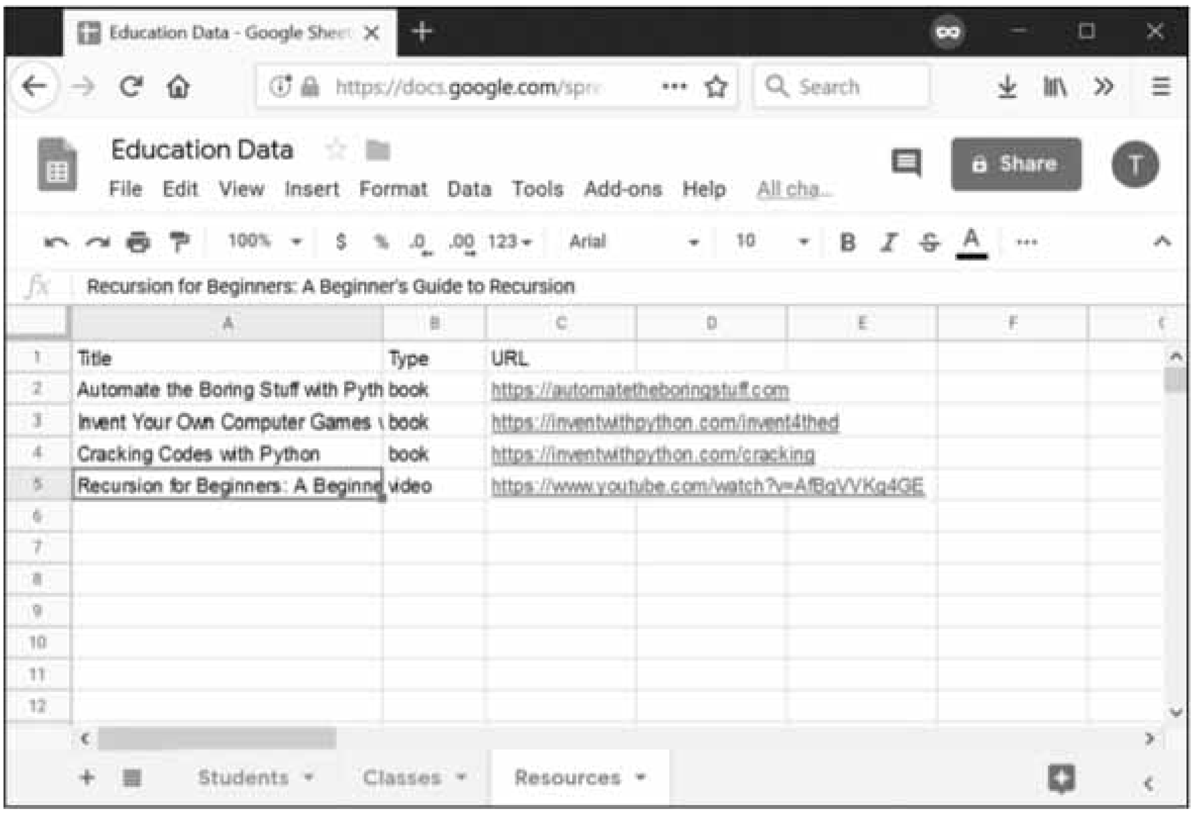


Figure 14-3: The Credentials page in the Google Cloud Platform developer's console

To generate a new credentials file from this page, click the **Create Credentials** button and select **OAuth client ID**, also shown in Figure 14-3. Next, for Application Type, select **Other** and give the file any name you like. This new credentials file will then be listed on the page, and you can click on the download icon to download it. The downloaded file will have a long, complicated filename, so you should rename it to the default filename that EZSheets attempts to load: *credentials-sheets.json*. You can also generate a new credential file by clicking the Enable the Google Sheets API button mentioned in the previous section.

## Spreadsheet Objects

In Google Sheets, a *spreadsheet* can contain multiple *sheets* (also called *worksheets*), and each sheet contains columns and rows of values. Figure 14-4 shows a spreadsheet titled "Education Data" containing three sheets titled "Students," "Classes," and "Resources." The first column of each sheet is labeled A, and the first row is labeled 1.


Education Data - Google Sheet
https://docs.google.com/spr
Search
Education Data
Share
File Edit View Insert Format Data Tools Add-ons Help All cha...
100% $ % .0 .00 123 Arial 10 B I S A
Students
Classes
Resources

fx Recursion for Beginners: A Beginner's Guide to Recursion

| | A | B | C |
|---|---|---|---|
| 1 | Title | Type | URL |
| 2 | Automate the Boring Stuff with Pyth | book | https://automatetheboringstuff.com |
| 3 | Invent Your Own Computer Games \ | book | https://inventwithpython.com/invent4thed |
| 4 | Cracking Codes with Python | book | https://inventwithpython.com/cracking |
| 5 | Recursion for Beginners: A Beginne | video | https://www.youtube.com/watch?v=AfBqVVKg4GE |

*Figure 14-4: A spreadsheet titled "Education Data" with three sheets*

While most of your work will involve modifying the `Sheet` objects, you can also modify `Spreadsheet` objects, as you'll see in the next section.

## *Creating, Uploading, and Listing Spreadsheets*

You can make a new `Spreadsheet` object from an existing spreadsheet, a blank spreadsheet, or an uploaded spreadsheet. To make a `Spreadsheet` object from an existing Google Sheets spreadsheet, you'll need the spreadsheet's ID string. The unique ID for a Google Sheets spreadsheet can be found in the URL, after the *spreadsheets/d/* part and before the */edit* part. For example, the spreadsheet featured in Figure 14-4 is located at the URL *https://docs.google.com/spreadsheets/d/1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU/edit#gid=151537240/*, so its ID is `1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU`.

**NOTE**

*The specific spreadsheet IDs used in this chapter are for my Google account's spreadsheets. They won't work if you enter them into your interactive shell. Go to* https://sheets.google.com/ *to create spreadsheets under your account and then get the IDs from the address bar.*

Pass your spreadsheet's ID as a string to the `ezsheets.Spreadsheet()` function to obtain a `Spreadsheet` object for its spreadsheet:

```
>>> import ezsheets
>>> ss = ezsheets.Spreadsheet('1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU')
>>> ss
Spreadsheet(spreadsheetId='1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU')
>>> ss.title
'Education Data'
```

For convenience, you can also obtain a `Spreadsheet` object of an existing spreadsheet by passing the spreadsheet's full URL to the function. Or, if there is only one spreadsheet in your Google account with that title, you can pass the title of the spreadsheet as a string.

To make a new, blank spreadsheet, call the `ezsheets.createSpreadsheet()` function and pass it a string for the new spreadsheet's title. For example, enter the following into the interactive shell:

```
>>> import ezsheets
>>> ss = ezsheets.createSpreadsheet('Title of My New Spreadsheet')
>>> ss.title
'Title of My New Spreadsheet'
```

To upload an existing Excel, OpenOffice, CSV, or TSV spreadsheet to Google Sheets, pass the filename of the spreadsheet to `ezsheets.upload()`. Enter the following into the interactive shell, replacing *my_spreadsheet.xlsx* with a spreadsheet file of your own:

```
>>> import ezsheets
>>> ss = ezsheets.upload('my_spreadsheet.xlsx')
>>> ss.title
'my_spreadsheet'
```

You can list the spreadsheets in your Google account by calling the `listSpreadsheets()` function. Enter the following into the interactive shell after uploading a spreadsheet:

```
>>> ezsheets.listSpreadsheets()
{'1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU': 'Education Data'}
```

The `listSpreadsheets()` function returns a dictionary where the keys are spreadsheet IDs and the values are the titles of each spreadsheet.

Once you've obtained a `Spreadsheet` object, you can use its attributes and methods to manipulate the online spreadsheet hosted on Google Sheets.

## *Spreadsheet Attributes*

While the actual data lives in a spreadsheet's individual sheets, the `Spreadsheet` object has the following attributes for manipulating the spreadsheet itself: `title`, `spreadsheetId`, `url`, `sheetTitles`, and `sheets`. Enter the following into the interactive shell:

```
>>> import ezsheets
>>> ss = ezsheets.Spreadsheet('1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU')
>>> ss.title         # The title of the spreadsheet.
'Education Data'
>>> ss.title = 'Class Data' # Change the title.
>>> ss.spreadsheetId # The unique ID (this is a read-only attribute).
'1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU'
>>> ss.url           # The original URL (this is a read-only attribute).
'https://docs.google.com/spreadsheets/d/1J-Jx6Ne2K_vqI9J2SO-
TAXOFbxx_9tUjwnkPC22LjeU/'
>>> ss.sheetTitles     # The titles of all the Sheet objects
('Students', 'Classes', 'Resources')
>>> ss.sheets          # The Sheet objects in this Spreadsheet, in order.
(<Sheet sheetId=0, title='Students', rowCount=1000, columnCount=26>, <Sheet
sheetId=1669384683, title='Classes', rowCount=1000, columnCount=26>, <Sheet
sheetId=151537240, title='Resources', rowCount=1000, columnCount=26>)
>>> ss[0]              # The first Sheet object in this Spreadsheet.
<Sheet sheetId=0, title='Students', rowCount=1000, columnCount=26>
>>> ss['Students']     # Sheets can also be accessed by title.
<Sheet sheetId=0, title='Students', rowCount=1000, columnCount=26>
>>> del ss[0]          # Delete the first Sheet object in this Spreadsheet.
>>> ss.sheetTitles     # The "Students" Sheet object has been deleted:
('Classes', 'Resources')
```

If someone changes the spreadsheet through the Google Sheets website, your script can update the `Spreadsheet` object to match the online data by calling the `refresh()` method:

```
>>> ss.refresh()
```

This will refresh not only the `Spreadsheet` object's attributes but also the data in the `Sheet` objects it contains. The changes you make to the `Spreadsheet` object will be reflected in the online spreadsheet in real time.

## *Downloading and Uploading Spreadsheets*

You can download a Google Sheets spreadsheet in a number of formats: Excel, OpenOffice, CSV, TSV, and PDF. You can also download it as a ZIP file containing HTML files of the spreadsheet's data. EZSheets contains functions for each of these options:

```
>>> import ezsheets
>>> ss = ezsheets.Spreadsheet('1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU')
>>> ss.title
'Class Data'
>>> ss.downloadAsExcel() # Downloads the spreadsheet as an Excel file.
'Class_Data.xlsx'
>>> ss.downloadAsODS() # Downloads the spreadsheet as an OpenOffice file.
'Class_Data.ods'
>>> ss.downloadAsCSV() # Only downloads the first sheet as a CSV file.
'Class_Data.csv'
>>> ss.downloadAsTSV() # Only downloads the first sheet as a TSV file.
'Class_Data.tsv'
>>> ss.downloadAsPDF() # Downloads the spreadsheet as a PDF.
'Class_Data.pdf'
>>> ss.downloadAsHTML() # Downloads the spreadsheet as a ZIP of HTML files.
'Class_Data.zip'
```

Note that files in the CSV and TSV formats can contain only one sheet; therefore, if you download a Google Sheets spreadsheet in this format, you will get the first sheet only. To download other sheets, you'll need to change the `Sheet` object's `index` attribute to `0`. See "Creating and Deleting Sheets" on page 341 for information on how to do this.

The download functions all return a string of the downloaded file's filename. You can also specify your own filename for the spreadsheet by passing the new filename to the download function:

```
>>> ss.downloadAsExcel('a_different_filename.xlsx')
'a_different_filename.xlsx'
```

The function should return the updated filename.

## *Deleting Spreadsheets*

To delete a spreadsheet, call the `delete()` method:

```
>>> import ezsheets
>>> ss = ezsheets.createSpreadsheet('Delete me') # Create the spreadsheet.
>>> ezsheets.listSpreadsheets() # Confirm that we've created a spreadsheet.
{'1aCw2NNJSZblDbhygVv77kPsL3djmgV5zJZllSOZ_mRk': 'Delete me'}
>>> ss.delete() # Delete the spreadsheet.
>>> ezsheets.listSpreadsheets()
{}
```

The `delete()` method will move your spreadsheet to the Trash folder on your Google Drive. You can view the contents of your Trash folder at *https://drive.google.com/drive/trash*. To permanently delete your spreadsheet, pass `True` for the `permanent` keyword argument:

```
>>> ss.delete(permanent=True)
```

In general, permanently deleting your spreadsheets is not a good idea, because it would be impossible to recover a spreadsheet that a bug in your script accidentally deleted. Even free Google Drive accounts have gigabytes of storage available, so you most likely don't need to worry about freeing up space.

## SHEET OBJECTS

A `Spreadsheet` object will have one or more `Sheet` objects. The `Sheet` objects represent the rows and columns of data in each sheet. You can access these sheets using the square brackets operator and an integer index. The `Spreadsheet` object's `sheets` attribute holds a tuple of `Sheet` objects in the order in which they appear in the spreadsheet. To access the `Sheet` objects in a spreadsheet, enter the following into the interactive shell:

```
>>> import ezsheets
>>> ss = ezsheets.Spreadsheet('1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU')
>>> ss.sheets    # The Sheet objects in this Spreadsheet, in order.
(<Sheet sheetId=1669384683, title='Classes', rowCount=1000, columnCount=26>,
<Sheet sheetId=151537240, title='Resources', rowCount=1000, columnCount=26>)
>>> ss.sheets[0] # Gets the first Sheet object in this Spreadsheet.
<Sheet sheetId=1669384683, title='Classes', rowCount=1000, columnCount=26>
>>> ss[0]        # Also gets the first Sheet object in this Spreadsheet.
<Sheet sheetId=1669384683, title='Classes', rowCount=1000, columnCount=26>
```

You can also obtain a `Sheet` object with the square brackets operator and a string of the sheet's name. The `Spreadsheet` object's `sheetTitles` attribute holds a tuple of all the sheet titles. For example, enter the following into the interactive shell:

```
>>> ss.sheetTitles # The titles of all the Sheet objects in this Spreadsheet.
('Classes', 'Resources')
>>> ss['Classes'] # Sheets can also be accessed by title.
<Sheet sheetId=1669384683, title='Classes', rowCount=1000, columnCount=26>
```

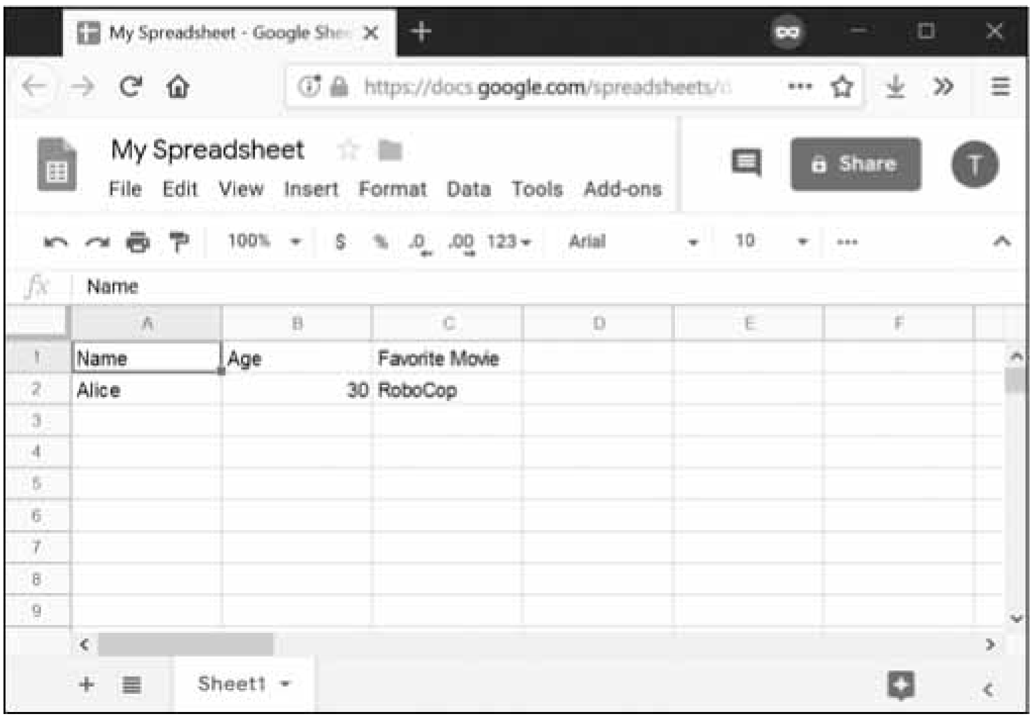
Once you have a `Sheet` object, you can read data from and write data to it using the `Sheet` object's methods, as explained in the next section.

## *Reading and Writing Data*

Just as in Excel, Google Sheets worksheets have columns and rows of cells containing data. You can use the square brackets operator to read and write data from and to these cells. For example, to create a new spreadsheet and add data to it, enter the following into the interactive shell:

```
>>> import ezsheets
>>> ss = ezsheets.createSpreadsheet('My Spreadsheet')
>>> sheet = ss[0] # Get the first sheet in this spreadsheet.
>>> sheet.title
'Sheet1'
>>> sheet = ss[0]
>>> sheet['A1'] = 'Name' # Set the value in cell A1.
>>> sheet['B1'] = 'Age'
>>> sheet['C1'] = 'Favorite Movie'
>>> sheet['A1'] # Read the value in cell A1.
'Name'
>>> sheet['A2'] # Empty cells return a blank string.
''
>>> sheet[2, 1] # Column 2, Row 1 is the same address as B1.
'Age'
>>> sheet['A2'] = 'Alice'
>>> sheet['B2'] = 30
>>> sheet['C2'] = 'RoboCop'
```

These instructions should produce a Google Sheets spreadsheet that looks like Figure 14-5.


My Spreadsheet - Google Sheets
https://docs.google.com/spreadsheets/
My Spreadsheet
File Edit View Insert Format Data Tools Add-ons
Share
100% $ % .0 .00 123 Arial 10
Sheet1

|   | A | B | C |
|---|---|---|---|
| 1 | Name | Age | Favorite Movie |
| 2 | Alice | 30 | RoboCop |

*Figure 14-5: The spreadsheet created with the example instructions*

Multiple users can update a sheet simultaneously. To refresh the local data in the `Sheet` object, call its `refresh()` method:

```
>>> sheet.refresh()
```

All of the data in the `Sheet` object is loaded when the `Spreadsheet` object is first loaded, so the data is read instantly. However, writing values to the online spreadsheet requires a network connection and can take about a second. If you have thousands of cells to update, updating them one at a time might be quite slow.

## Column and Row Addressing

Cell addressing works in Google Sheets just like in Excel. The only difference is that, unlike Python's 0-based list indexes, Google Sheets have 1-based columns and rows: the first column or row is at index 1, not 0. You can convert the `'A2'` string-style address to the `(column, row)` tuple-style address (and vice versa) with the `convertAddress()` function. The `getColumnLetterOf()` and `getColumnNumberOf()` functions will also convert a column address between letters and numbers. Enter the following into the interactive shell:

```
>>> import ezsheets
>>> ezsheets.convertAddress('A2') # Converts addresses...
(1, 2)
>>> ezsheets.convertAddress(1, 2) # ...and converts them back, too.
'A2'
>>> ezsheets.getColumnLetterOf(2)
'B'
```

```
>>> ezsheets.getColumnNumberOf('B')
2
>>> ezsheets.getColumnLetterOf(999)
'ALK'
>>> ezsheets.getColumnNumberOf('ZZZ')
18278
```

The `'A2'` string-style addresses are convenient if you're typing addresses into your source code. But the `(column, row)` tuple-style addresses are convenient if you're looping over a range of addresses and need a numeric form for the column. The `convertAddress()`, `getColumnLetterOf()`, and `getColumnNumberOf()` functions are helpful when you need to convert between the two formats.

## Reading and Writing Entire Columns and Rows

As mentioned, writing data one cell at a time can often take too long. Fortunately, EZSheets has `Sheet` methods for reading and writing entire columns and rows at the same time. The `getColumn()`, `getRow()`, `updateColumn()`, and `updateRow()` methods will, respectively, read and write columns and rows. These methods make requests to the Google Sheets servers to update the spreadsheet, so they require that you be connected to the internet. In this section's example, we'll upload *produceSales.xlsx* from the last chapter to Google Sheets. The first eight rows look like Table 14-1.

**Table 14-1:** The First Eight Rows of the *produceSales.xlsx* Spreadsheet

| | A | B | C | D |
|---|---|---|---|---|
| 1 | PRODUCE | COST PER POUND | POUNDS SOLD | TOTAL |
| 2 | Potatoes | 0.86 | 21.6 | 18.58 |
| 3 | Okra | 2.26 | 38.6 | 87.24 |
| 4 | Fava beans | 2.69 | 32.8 | 88.23 |
| 5 | Watermelon | 0.66 | 27.3 | 18.02 |
| 6 | Garlic | 1.19 | 4.9 | 5.83 |
| 7 | Parsnips | 2.27 | 1.1 | 2.5 |
| 8 | Asparagus | 2.49 | 37.9 | 94.37 |

To upload this spreadsheet, enter the following into the interactive shell:

```
>>> import ezsheets
>>> ss = ezsheets.upload('produceSales.xlsx')
>>> sheet = ss[0]
>>> sheet.getRow(1) # The first row is row 1, not row 0.
['PRODUCE', 'COST PER POUND', 'POUNDS SOLD', 'TOTAL', '', '']
>>> sheet.getRow(2)
['Potatoes', '0.86', '21.6', '18.58', '', '']
>>> columnOne = sheet.getColumn(1)
>>> sheet.getColumn(1)
['PRODUCE', 'Potatoes', 'Okra', 'Fava beans', 'Watermelon', 'Garlic',
--snip--
>>> sheet.getColumn('A') # Same result as getColumn(1)
['PRODUCE', 'Potatoes', 'Okra', 'Fava beans', 'Watermelon', 'Garlic',
--snip--
>>> sheet.getRow(3)
['Okra', '2.26', '38.6', '87.24', '', '']
>>> sheet.updateRow(3, ['Pumpkin', '11.50', '20', '230'])
>>> sheet.getRow(3)
['Pumpkin', '11.50', '20', '230', '', '']
>>> columnOne = sheet.getColumn(1)
>>> for i, value in enumerate(columnOne):
...     # Make the Python list contain uppercase strings:
...     columnOne[i] = value.upper()
...
>>> sheet.updateColumn(1, columnOne) # Update the entire column in one
request.
```

The `getRow()` and `getColumn()` functions retrieve the data from every cell in a specific row or column as a list of values. Note that empty cells become blank string values in the list. You can pass `getColumn()` either a column number or letter to tell it to retrieve a specific column's data. The previous example shows that `getColumn(1)` and `getColumn('A')` return the same list.

The `updateRow()` and `updateColumn()` functions will overwrite all the data in the row or column, respectively, with the list of values passed to the function. In this example, the third row initially contains information about okra, but the `updateRow()` call replaces it with data about pumpkin. Call `sheet.getRow(3)` again to view the new values in the third row.

Next, let's update the "produceSales" spreadsheet. Updating cells one at a time is slow if you have many cells to update. Getting a column or row as a list, updating the list, and then updating the entire column or row with the list is much faster, since all the changes can be made in one request.

To get all of the rows at once, call the `getRows()` method to return a list of lists. The inner lists inside the outer list each represent a single row of the sheet. You can modify the values in this data structure to change the produce name, pounds sold, and total cost of some of the rows. Then you pass it to the `updateRows()` method by entering the following into the interactive shell:

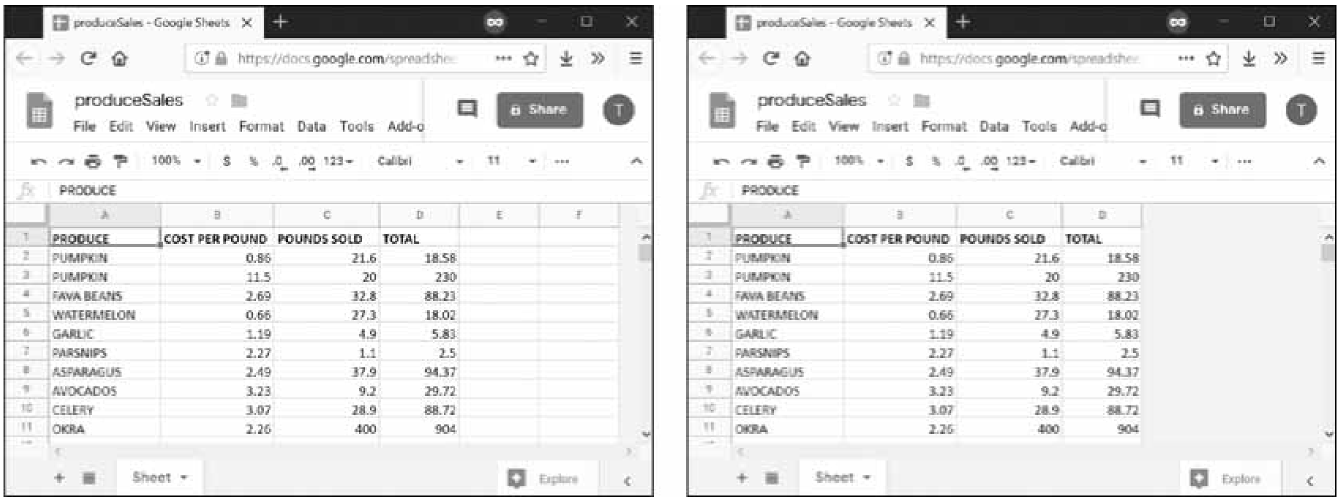
```
>>> rows = sheet.getRows() # Get every row in the spreadsheet.
>>> rows[0] # Examine the values in the first row.
['PRODUCE', 'COST PER POUND', 'POUNDS SOLD', 'TOTAL', '', '']
>>> rows[1]
['POTATOES', '0.86', '21.6', '18.58', '', '']
>>> rows[1][0] = 'PUMPKIN' # Change the produce name.
>>> rows[1]
['PUMPKIN', '0.86', '21.6', '18.58', '', '']
>>> rows[10]
['OKRA', '2.26', '40', '90.4', '', '']
>>> rows[10][2] = '400' # Change the pounds sold.
>>> rows[10][3] = '904' # Change the total.
>>> rows[10]
['OKRA', '2.26', '400', '904', '', '']
>>> sheet.updateRows(rows) # Update the online spreadsheet with the changes.
```

You can update the entire sheet in a single request by passing `updateRows()` the list of lists returned from `getRows()`, amended with the changes made to rows 1 and 10.

Note that the rows in the Google Sheet have empty strings at the end. This is because the uploaded sheet has a column count of `6`, but we have only 4 columns of data. You can read the number of rows and columns in a sheet with the `rowCount` and `columnCount` attributes. Then by setting these values, you can change the size of the sheet.

```
>>> sheet.rowCount        # The number of rows in the sheet.
23758
>>> sheet.columnCount     # The number of columns in the sheet.
6
>>> sheet.columnCount = 4 # Change the number of columns to 4.
>>> sheet.columnCount     # Now the number of columns in the sheet is 4.
4
```

These instructions should delete the fifth and sixth columns of the “produceSales” spreadsheet, as shown in Figure 14-6.


produceSales - Google Sheets
https://docs.google.com/spreadshe
produceSales
File Edit View Insert Format Data Tools Add-o
Share
Sheet
Explore

|   | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | PRODUCE | COST PER POUND | POUNDS SOLD | TOTAL | | |
| 2 | PUMPKIN | 0.86 | 21.6 | 18.58 | | |
| 3 | PUMPKIN | 11.5 | 20 | 230 | | |
| 4 | FAVA BEANS | 2.69 | 32.8 | 88.23 | | |
| 5 | WATERMELON | 0.66 | 27.3 | 18.02 | | |
| 6 | GARLIC | 1.19 | 4.9 | 5.83 | | |
| 7 | PARSNIPS | 2.27 | 1.1 | 2.5 | | |
| 8 | ASPARAGUS | 2.49 | 37.9 | 94.37 | | |
| 9 | AVOCADOS | 3.23 | 9.2 | 29.72 | | |
| 10 | CELERY | 3.07 | 28.9 | 88.72 | | |
| 11 | OKRA | 2.26 | 400 | 904 | | |

|   | A | B | C | D |
|---|---|---|---|---|
| 1 | PRODUCE | COST PER POUND | POUNDS SOLD | TOTAL |
| 2 | PUMPKIN | 0.86 | 21.6 | 18.58 |
| 3 | PUMPKIN | 11.5 | 20 | 230 |
| 4 | FAVA BEANS | 2.69 | 32.8 | 88.23 |
| 5 | WATERMELON | 0.66 | 27.3 | 18.02 |
| 6 | GARLIC | 1.19 | 4.9 | 5.83 |
| 7 | PARSNIPS | 2.27 | 1.1 | 2.5 |
| 8 | ASPARAGUS | 2.49 | 37.9 | 94.37 |
| 9 | AVOCADOS | 3.23 | 9.2 | 29.72 |
| 10 | CELERY | 3.07 | 28.9 | 88.72 |
| 11 | OKRA | 2.26 | 400 | 904 |

*Figure 14-6: The sheet before (left) and after (right) changing the column count to 4*

According to *https://support.google.com/drive/answer/37603?hl=en/*, Google Sheets spreadsheets can have up to 5 million cells in them. However, it’s a good idea to make sheets only as big as you need to minimize the time it takes to update and refresh the data.

## *Creating and Deleting Sheets*

All Google Sheets spreadsheets start with a single sheet named “Sheet1.” You can add additional sheets to the end of the list of sheets with the `createSheet()` method, to which you pass a string to use as the new sheet’s title. An optional second argument can specify the integer index of the new sheet. To create a spreadsheet and then add new sheets to it, enter the following into the interactive shell:

```
>>> import ezsheets
>>> ss = ezsheets.createSpreadsheet('Multiple Sheets')
>>> ss.sheetTitles
('Sheet1',)
>>> ss.createSheet('Spam') # Create a new sheet at the end of the list of
sheets.
<Sheet sheetId=2032744541, title='Spam', rowCount=1000, columnCount=26>
>>> ss.createSheet('Eggs') # Create another new sheet.
<Sheet sheetId=417452987, title='Eggs', rowCount=1000, columnCount=26>
>>> ss.sheetTitles
('Sheet1', 'Spam', 'Eggs')
>>> ss.createSheet('Bacon', 0) code># Create a sheet at index 0 in the list of
sheets.
```

```
<Sheet sheetId=814694991, title='Bacon', rowCount=1000, columnCount=26>
>>> ss.sheetTitles
('Bacon', 'Sheet1', 'Spam', 'Eggs')
```

These instructions add three new sheets to the spreadsheet: “Bacon,” “Spam,” and “Eggs” (in addition to the default “Sheet1”). The sheets in a spreadsheet are ordered, and new sheets go to the end of the list unless you pass a second argument to `createSheet()` specifying the sheet’s index. Here, you create the sheet titled “Bacon” at index `0`, making “Bacon” the first sheet in the spreadsheet and displacing the other three sheets by one position. This is similar to the behavior of the `insert()` list method.

You can see the new sheets on the tabs at the bottom of the screen, as shown in Figure 14-7.

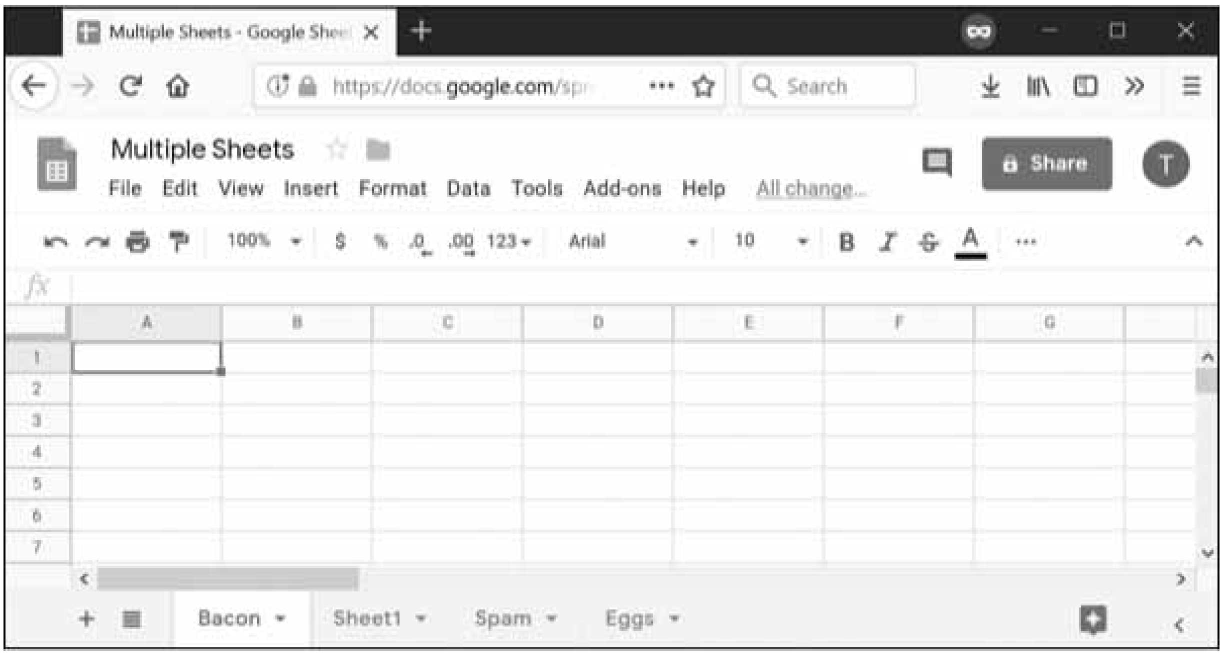

Multiple Sheets - Google Sheets
https://docs.google.com/sp
Search
Multiple Sheets
Share
File Edit View Insert Format Data Tools Add-ons Help All change...
100%
Arial
10
Bacon
Sheet1
Spam
Eggs

*Figure 14-7: The “Multiple Sheets” spreadsheet after adding sheets “Spam,” “Eggs,” and “Bacon”*

The `Sheet` object’s `delete()` method will delete the sheet from the spreadsheet. If you want to keep the sheet but delete the data it contains, call the `clear()` method to clear all the cells and make it a blank sheet. Enter the following into the interactive shell:

```
>>> ss.sheetTitles
('Bacon', 'Sheet1', 'Spam', 'Eggs')
>>> ss[0].delete()     # Delete the sheet at index 0: the "Bacon" sheet.
>>> ss.sheetTitles
('Sheet1', 'Spam', 'Eggs')
>>> ss['Spam'].delete() # Delete the "Spam" sheet.
>>> ss.sheetTitles
('Sheet1', 'Eggs')
>>> sheet = ss['Eggs']  # Assign a variable to the "Eggs" sheet.
>>> sheet.delete()      # Delete the "Eggs" sheet.
>>> ss.sheetTitles
('Sheet1',)
>>> ss[0].clear()       # Clear all the cells on the "Sheet1" sheet.
>>> ss.sheetTitles      # The "Sheet1" sheet is empty but still exists.
('Sheet1',)
```

Deleting sheets is permanent; there's no way to recover the data. However, you can back up sheets by copying them to another spreadsheet with the `copyTo()` method, as explained in the next section.

### *Copying Sheets*

Every `Spreadsheet` object has an ordered list of the `Sheet` objects it contains, and you can use this list to reorder the sheets (as shown in the previous section) or copy them to other spreadsheets. To copy a `Sheet` object to another `Spreadsheet` object, call the `copyTo()` method. Pass it the destination `Spreadsheet` object as an argument. To create two spreadsheets and copy the first spreadsheet's data to the other sheet, enter the following into the interactive shell:

```
>>> import ezsheets
>>> ss1 = ezsheets.createSpreadsheet('First Spreadsheet')
>>> ss2 = ezsheets.createSpreadsheet('Second Spreadsheet')
>>> ss1[0]
<Sheet sheetId=0, title='Sheet1', rowCount=1000, columnCount=26>
>>> ss1[0].updateRow(1, ['Some', 'data', 'in', 'the', 'first', 'row'])
>>> ss1[0].copyTo(ss2) # Copy the ss1's Sheet1 to the ss2 spreadsheet.
>>> ss2.sheetTitles    # ss2 now contains a copy of ss1's Sheet1.
('Sheet1', 'Copy of Sheet1')
```

Note that since the destination spreadsheet (`ss2` in the previous example) already had a sheet named `Sheet1`, the copied sheet will be named `Copy of Sheet1`. Copied sheets appear at the end of the list of the destination spreadsheet's sheets. If you wish, you can change their `index` attribute to reorder them in the new spreadsheet.

## WORKING WITH GOOGLE SHEETS QUOTAS

Because Google Sheets is online, it's easy to share sheets among multiple users who can all access the sheets simultaneously. However, this also means that reading and updating the sheets will be slower than reading and updating Excel files stored locally on your hard drive. In addition, Google Sheets has limits on how many read and write operations you can perform.

According to Google's developer guidelines, users are restricted to creating 250 new spreadsheets a day, and free Google accounts can perform 100 read and 100 write requests per 100 seconds. Attempting to exceed this quota will raise the `googleapiclient.errors.HttpError` "Quota exceeded for quota group" exception. EZSheets will automatically catch this exception and retry the request. When this happens, the function calls to read or write data will take several seconds (or even a full minute or two) before they return. If the request continues to fail (which is possible if another script using the same credentials is also making requests), EZSheets will re-raise this exception.

This means that, on occasion, your EZSheets method calls may take several seconds before they return. If you want to view your API usage or increase your quota, go to the IAM & Admin Quotas page at *https://console.developers.google.com/quotas/* to learn about paying for increased usage. If you'd rather just deal with the `HttpError` exceptions yourself, you can set `ezsheets.IGNORE_QUOTA` to `True`, and EZSheet's methods will raise these exceptions when it encounters them.

## Summary

Google Sheets is a popular online spreadsheet application that runs in your browser. Using the EZSheets third-party module, you can download, create, read, and modify spreadsheets. EZSheets represents spreadsheets as `Spreadsheet` objects, each of which contains an ordered list of `Sheet` objects. Each sheet has columns and rows of data that you can read and update in several ways.

While Google Sheets makes sharing data and cooperative editing easy, its main disadvantage is speed: you must update spreadsheets with web requests, which can take a few seconds to execute. But for most purposes, this speed restriction won't affect Python scripts using EZSheets. Google Sheets also limits how often you can make changes.

For complete documentation of EZSheet's features, visit *https://ezsheets.readthedocs.io/*.

## Practice Questions

1. What three files do you need for EZSheets to access Google Sheets?
2. What two types of objects does EZSheets have?
3. How can you create an Excel file from a Google Sheet spreadsheet?

4. How can you create a Google Sheet spreadsheet from an Excel file?
5. The `ss` variable contains a `Spreadsheet` object. What code will read data from the cell B2 in a sheet titled “Students”?
6. How can you find the column letters for column 999?
7. How can you find out how many rows and columns a sheet has?
8. How do you delete a spreadsheet? Is this deletion permanent?
9. What functions will create a new `Spreadsheet` object and a new `Sheet` object, respectively?
10. What will happen if, by making frequent read and write requests with EZSheets, you exceed your Google account’s quota?

## Practice Projects

For practice, write programs to do the following tasks.

### *Downloading Google Forms Data*

Google Forms allows you to create simple online forms that make it easy to collect information from people. The information they enter into the form is stored in a Google Sheet. For this project, write a program that can automatically download the form information that users have submitted. Go to *https://docs.google.com/forms/* and start a new form; it will be blank. Add fields to the form that ask the user for a name and email address. Then click the **Send** button in the upper right to get a link to your new form, such as *https://goo.gl/forms/QZsq5sC2Qe4fYO592/*. Try to enter a few example responses into this form.

On the “Responses” tab of your form, click the green **Create Spreadsheet** button to create a Google Sheets spreadsheet that will hold the responses that users submit. You should see your example responses in the first rows of this spreadsheet. Then write a Python script using EZSheets to collect a list of the email addresses on this spreadsheet.

### *Converting Spreadsheets to Other Formats*

You can use Google Sheets to convert a spreadsheet file into other formats. Write a script that passes a submitted file to `upload()`. Once the spreadsheet has uploaded to Google Sheets, download it using `downloadAsExcel()`, `downloadAsODS()`, and other such functions to create a copy of the spreadsheet in these other formats.

### *Finding Mistakes in a Spreadsheet*

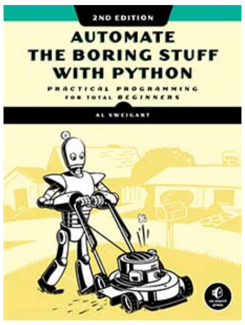
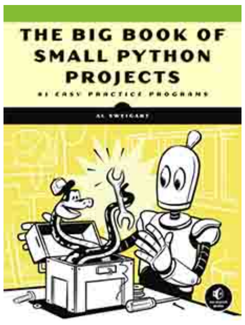
After a long day at the bean-counting office, I've finished a spreadsheet with all the bean totals and uploaded them to Google Sheets. The spreadsheet is publicly viewable (but not editable). You can get this spreadsheet with the following code:

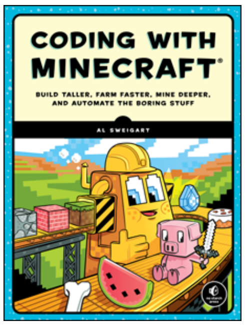
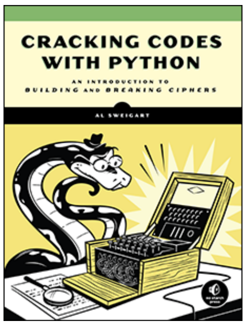
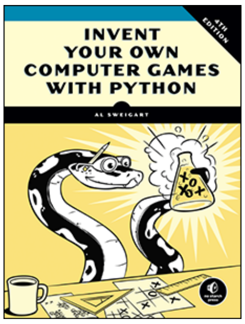
```
>>> import ezsheets
>>> ss = ezsheets.Spreadsheet('1jDZEdvSIh4TmZxccyy0ZXrH-ELlrwq8_YYiZrEOB4jg')
```

You can look at this spreadsheet in your browser by going to *https://docs.google.com/spreadsheets/d/1jDZEdvSIh4TmZxccyy0ZXrH-ELlrwq8_YYiZrEOB4jg/edit?usp=sharing/*. The columns of the first sheet in this spreadsheet are "Beans per Jar," "Jars," and "Total Beans." The "Total Beans" column is the product of the numbers in the "Beans per Jar" and "Jars" columns. However, there is a mistake in one of the 15,000 rows in this sheet. That's too many rows to check by hand. Luckily, you can write a script that checks the totals.

As a hint, you can access the individual cells in a row with `ss[0].getRow(`*`rowNum`*`)`, where *`ss`* is the `Spreadsheet` object and *`rowNum`* is the row number. Remember that row numbers in Google Sheets begin at 1, not 0. The cell values will be strings, so you'll need to convert them to integers so your program can work with them. The expression `int(`*`ss`*`[0].getRow(2)[0]) * int(`*`ss`*`[0].getRow(2)[1]) == int(`*`ss`*`[0].getRow(2)[2])` evaluates to `True` if the row has the correct total. Put this code in a loop to identify which row in the sheet has the incorrect total.

ToC


2ND EDITION
AUTOMATE THE BORING STUFF WITH PYTHON
PRACTICAL PROGRAMMING FOR TOTAL BEGINNERS
AL SWEIGART

THE BIG BOOK OF SMALL PYTHON PROJECTS
81 EASY PRACTICE PROGRAMS
AL SWEIGART

BEYOND THE BASIC STUFF WITH PYTHON
BEST PRACTICES FOR WRITING CLEAN CODE
AL SWEIGART

CODING WITH MINECRAFT
BUILD TALLER, FARM FASTER, MINE DEEPER, AND AUTOMATE THE BORING STUFF
AL SWEIGART

CRACKING CODES WITH PYTHON
AN INTRODUCTION TO BUILDING AND BREAKING CIPHERS
AL SWEIGART


SCRATCH 3 PROGRAMMING PLAYGROUND
LEARN TO PROGRAM BY MAKING COOL GAMES
AL SWEIGART
COVERS SCRATCH 3

Making Games with Python & Pygame
A guide to programming with graphics, animation, and sound
Al Sweigart