

○○○

```
from pydub import AudioSegment

audio_file = AudioSegment.from_mp3("example.mp3")
louder_audio_file = audio_file + 18
louder_audio_file.export("example_louder.mp3", format="mp3")
```

GUIDES

13 min to read

## 12 useful Python scripts for developers

WRITTEN BY

Madhura Kumar

PUBLISHED ON

Jun 8, 2022

It's no secret that a developer's day-to-day work often involves creating and maintaining small utility scripts. These scripts are the glue that connects various aspects of your system or build environment. While these Python scripts may not be complex, maintaining them can become a tedious chore that could cost you time and money.

One way to lighten the maintenance load is to use script automation. Rather than spending your time running scripts (often manually), **script automation allows you to schedule these tasks to run on a particular timetable or be triggered in response to certain events.**

This article will cover twelve Python scripts that were selected for their general utility, ease of use, and positive impact on your workload. They range in complexity from easy to intermediate and focus on text processing and file management. Specifically, we'll walk through the following use cases:

1. Create strong random passwords

2. Extract text from a PDF
3. Text processing with Pandoc
4. Manipulate audio with Pydub
5. Filter text
6. Locate addresses
7. Convert a CSV to Excel
8. Pattern match with regular expressions
9. Convert images to JPG
10. Compress images
11. Get content from Wikipedia
12. Create and manage Heroku apps

These scripts can be dropped into just about any workflow or run as part of an automated playbook using a tool like [Airplane](#).

## Using Airplane to manage and execute Python scripts

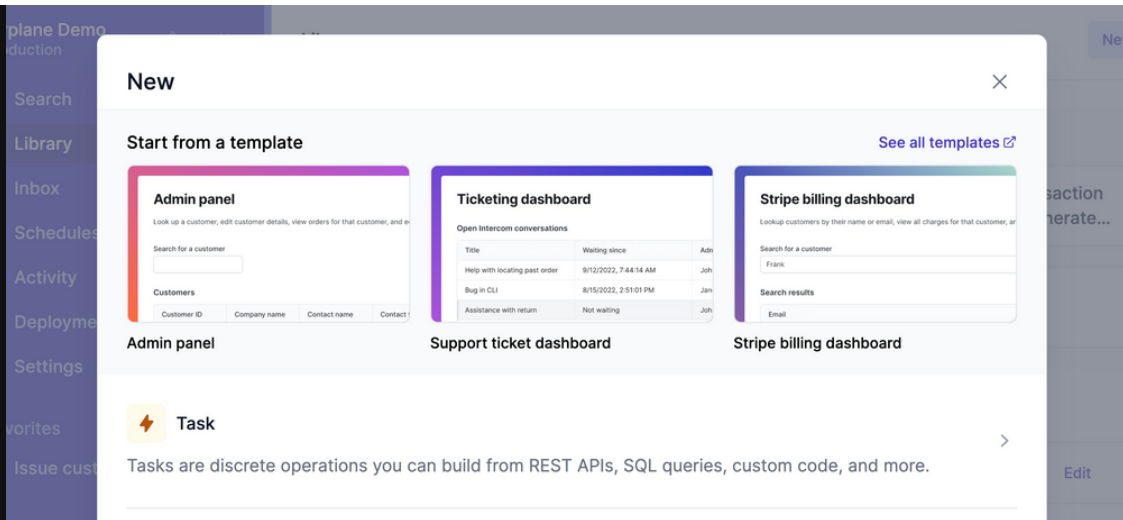
[Airplane](#) is a developer platform to transform APIs, SQL queries, and scripts into internal applications in minutes. The platform provides a central location to store your Python scripts as well as to run, manage, and share them securely.

You can also use Airplane [schedules](#) as a substitute for cron and other job schedulers and Airplane provides [permissions](#), [audit logs](#), [approval flows](#), and much more out of the box.

Before we jump into some useful Python scripts, let's quickly walk through how you can use Airplane to manage and execute them.

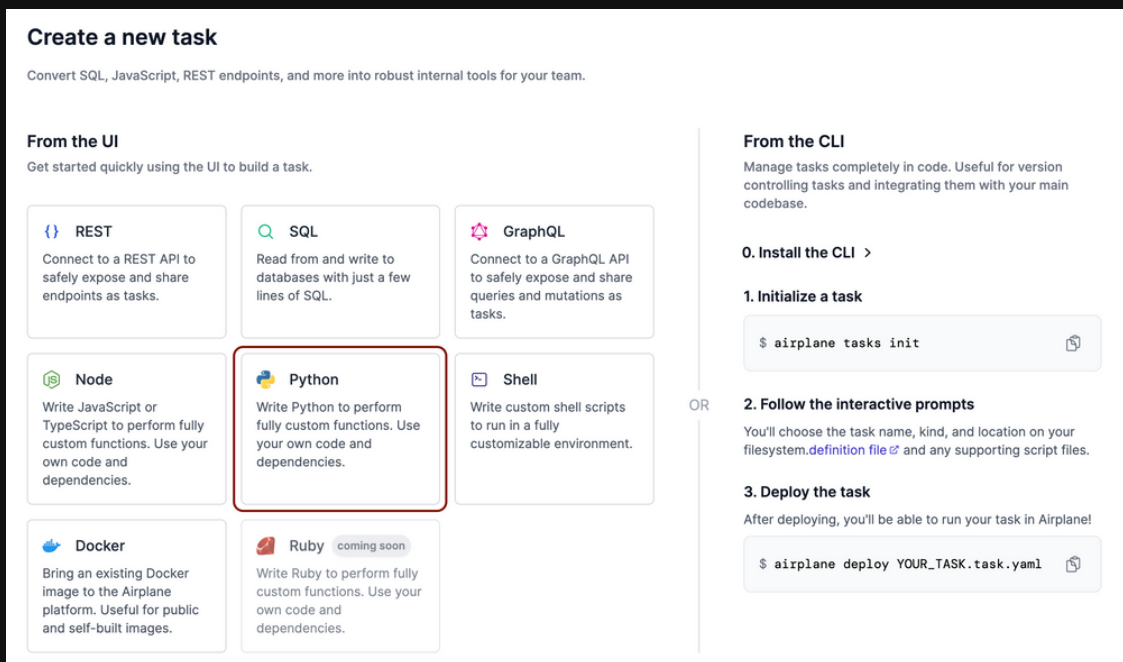
You can get started by [signing up for a free Airplane account](#) and heading to: **Library** > '+' to create a new [Task](#). A task represents a single step or operation such as 'hit X API endpoint' or 'run X script'. A task can be a SQL query, custom TypeScript/JavaScript/Python, or even a wrapper around a REST API call.

You can also create [Views](#) (custom UIs) or orchestrate [Workflows](#) (multi-step code-based operations) from this menu. Select **Task** as shown below:



A task can be a SQL query, custom TypeScript/JavaScript/Python, or even a wrapper around a REST API call.

There are a number of different options for creating a task. Let's select Python. Note that you can create your task from the UI as described below but you can also create and define your task fully in code from the CLI.



Add a task name and description and click **Continue**.

Python 2 Define 3 Build 4 Advanced

## Define your task

Name filter\_regex Folder No folder

Filter Regex

Description (optional)

A task to filter text using regular expressions

Parameters + Add parameter

Parameters define what users are asked to input into this task. [Docs](#)

< Back Continue >

We'll first create our task and then deploy our code. Click **Create task** to finish adding the task to Airplane.

Python Define 3 Build 4 Advanced

## Build your task

**i** Create the task, then deploy your code.  
After creating the task, you'll be able to use the [Airplane CLI](#) to write and deploy your code.

Environment Variables + Add variable

Resources + Attach resource

Attach resources to access them from within your task. [Docs](#)

**?** Need help? Visit the [documentation](#) for more details and examples.

< Back Continue > Create task

Next, we'll use the [Airplane CLI tool](#) to write a script for this task. This tool will allow you to create and manage scripts locally on your machine.

Be sure to wrap your code in a function called `main`. [Dependencies](#) can be included in a `requirements.txt` file saved in your script's parent directory.

Before deploying our Python script to Airplane, we'll create a `.yaml` file containing the name of the script so it's easily identifiable.

For example, if you are filtering text with regular expressions, you might title this file `regex_filter.task.yaml`. An explanation of the task definition can be found in the Airplane [task definition docs](#).

Your completed `.yaml` file should look similar to the example below:

```
python
1 slug: regex_filter
2 name: "Regex Filter"
3 description: Filter text using Regular Expressions
4 python:
5   entrypoint: regex_filter.py
```

Once your Python script and `.yaml` task definition file have been created, you can deploy to Airplane using the CLI: `airplane deploy regex_filter.task.yaml`

Your Python task is now ready to run!

You can find more details on getting started in the [Airplane developer docs for Python](#), [quickstart guide](#), and [guide to getting started with runbooks](#).

Now that we know how to use Airplane to manage and execute Python scripts, let's walk through some of the scripts themselves.

## Python scripts for developers to implement

Let's dive into twelve Python utility scripts that we can leverage to make our lives easier. The code samples from this article can be found in [this GitHub repository](#).

### 1. Create strong random passwords

There are many reasons to create strong random passwords, from onboarding new users to providing a password-reset workflow to creating a new password when rotating credentials. You can easily use a dependency-free Python script to automate this process:

```
python
1 # Generate Strong Random Passwords
2 import random
3 import string
4 # This script will generate an 18 character password
5 word_length = 18
6 # Generate a list of letters, digits, and some punctuation
7 components = [string.ascii_letters, string.digits, "!@#%&"]
8 # flatten the components into a list of characters
9 chars = []
10 for clist in components:
11     for item in clist:
12         chars.append(item)
13 def generate_password():
14     # Store the generated password
15     password = []
```

```
16 # Choose a random item from 'chars' and add it to 'password'
17 for i in range(word_length):
18     rchar = random.choice(chars)
19     password.append(rchar)
20 # Return the composed password as a string
21 return "".join(password)
22 # Output generated password
23 print(generate_password())
```

## 2. Extract text from a PDF

Python can also be used to easily extract text from PDFs using the `PyPDF2` package. Getting text from a PDF file proves useful for data mining, invoice reconciliation, or report generation, and the extraction process can be automated in just a few lines of code. You can run `pip install PyPDF2` in your terminal to install the package. Below are a few examples of what you can achieve using Py2PDF2:

Say you receive a multipage PDF file but you only need the first page. The script below allows you to extract text from the first page in a PDF with just a few lines of Python code:

```
python
1 # import module PyPDF2
2 import PyPDF2
3 # put 'example.pdf' in working directory
4 # and open it in read binary mode
5 pdfFileObj = open('example.pdf', 'rb')
6 # call and store PdfFileReader
7 # object in pdfReader
8 pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
9 # to print the total number of pages in pdf
10 # print(pdfReader.numPages)
11 # get specific page of pdf by passing
12 # number since it stores pages in list
13 # to access first page pass 0
14 pageObj = pdfReader.getPage(0)
15 # extract the page object
16 # by extractText() function
17 texts = pageObj.extractText()
18 # print the extracted texts
19 print(texts)
```

Maybe you'd like to copy text from two PDF files and merge the text into a new PDF. You can do this using the code below:

```
python
```

These examples, along with several others, can be found [here](#).

## 3. Text processing with Pandoc

[Pandoc](#) is a fully featured command-line tool that allows you to convert markup between different formats. This means you can use Pandoc to convert Markdown text directly to docx or MediaWiki markup to DocBook. Markup-format conversion allows you to process external content or user-

submitted information without restricting the data to a single format. You can install the `pandoc` package with `pip`. Following are a few examples of what you can do with `pandoc`.

First, say you receive a `markdown` formatted document but need to convert it to PDF. `pandoc` makes this easy:

```
python
1 import pandoc
2
3 in_file = open("example.md", "r").read()
4 pandoc.write(in_file, file="example.pdf", format="pdf")
```

Or maybe you'd like to convert the `markdown` file to a `json` object. You can use the following script to do so:

```
python
1 import pandoc
2 md_string = """
3 # Hello from Markdown
4
5 **This is a markdown string**
6 """
7 input_string = pandoc.read(md_string)
8 pandoc.write(input_string, format="json", file="md.json")
```

You can find many other [example functions here](#), or check out the `pandoc` [package documentation](#) for more information.

## 4. Manipulate audio with Pydub

`Pydub` is a Python package that allows you to manipulate audio, including converting audio to various file formats like `wav` or `mp3`. Additionally, `Pydub` can segment an audio file into millisecond samples, which may be particularly useful for machine learning tasks. `Pydub` can be installed by entering `pip install pydub` in your terminal.

Say you're working with audio and need to ensure each file has the proper volume. You can use this script to automate the task:

```
python
1 from pydub import AudioSegment
2
3 audio_file = AudioSegment.from_mp3("example.mp3")
4 louder_audio_file = audio_file + 18
5 louder_audio_file.export("example_louder.mp3", format="mp3")
```

`Pydub` has many additional features not covered in this example. You can find more of these in the [Pydub GitHub repository](#).

## 5. Filter text

Matching and filtering text with regular expressions in Python is simple, and the benefits can be enormous. Say you have a system for batch processing sales-confirmation messages, and you need to extract a credit card from the text of an email message. The script below can quickly find any credit card number that matches the pattern, allowing you to easily filter this information from any textual content:

```
python
1 # Filter Text
2 # Import re module
3 import re
4 # Take any string data
5 string = """a string we are using to filter specific items.
6 perhaps we would like to match credit card numbers
7 mistakenly entered into the user input. 4444 3232 1010 8989
8 and perhaps another? 9191 0232 9999 1111"""
9
10 # Define the searching pattern
11 pattern = '(([0-9](\s+)?)\d{4}){4}'
12
13 # match the pattern with input value
14 found = re.search(pattern, string)
15 print(found)
16 # Print message based on the return value
17 if found:
18     print("Found a credit card number!")
19 else:
20     print("No credit card numbers present in input")
```

## 6. Locate addresses

Locating an address can be useful if you're dealing with shipping or delivery logistics or for simple user-profiling tasks. To get started, install `geocoder` by running `pip install geocoder` in your terminal. The script below allows you to easily find the latitude and longitude coordinates for any address or to find an address from any set of coordinates:

```
python
1 import geocoder
2 address = "1600 Pennsylvania Ave NW, Washington DC USA"
3 coordinates = geocoder.arcgis(address)
4 geo = geocoder.arcgis(address)
5 print(geo.latlng)
6 # output: [38.89767510765125, -77.03654699820865]
7
8 # If we want to retrieve the location from a set of coordinates
9 # perform a reverse query.
10 location = geocoder.arcgis([38.89767510765125, -77.03654699820865], method="reverse")
11
12 # output: <[OK] Arcgis - Reverse [White House]>
13 print(location)
```

## 7. Convert a CSV to Excel

You may find yourself frequently managing CSV file outputs from an analytics platform or a dataset. Opening the CSV file in Excel is relatively simple, but Python allows you to skip this manual step by automating the conversion. This also allows you to manipulate the CSV data before conversion into Excel, saving additional time and effort.



Start by downloading the `openpyxl` package using `pip install openpyxl`. Once `openpyxl` is installed, you can use the script below to convert a CSV file to an Excel spreadsheet:

```
python
1 #!python3
2 # -*- coding: utf-8 -*-
3
4 import openpyxl
5 import sys
6
7 #inputs
8 print("This programme writes the data in any Comma-separated value file (such as: .csv or .data
9 print("The input and output files must be in the same directory of the python file for the prog
10
11 csv_name = input("Name of the CSV file for input (with the extension): ")
12 sep = input("Separator of the CSV file: ")
13 excel_name = input("Name of the excel file for output (with the extension): ")
14 sheet_name = input("Name of the excel sheet for output: ")
15
16 #opening the files
17 try:
18     wb = openpyxl.load_workbook(excel_name)
19     sheet = wb.get_sheet_by_name(sheet_name)
20
21     file = open(csv_name,"r",encoding = "utf-8")
22 except:
23     print("File Error!")
24     sys.exit()
25
26 #rows and columns
27 row = 1
28 column = 1
29
30 #for each line in the file
31 for line in file:
32     #remove the \n from the line and make it a list with the separator
33     line = line[:-1]
34     line = line.split(sep)
35
36     #for each data in the line
37     for data in line:
38         #write the data to the cell
39         sheet.cell(row,column).value = data
40         #after each data column number increases by 1
41         column += 1
42
43     #to write the next line column number is set to 1 and row number is increased by 1
44     column = 1
45     row += 1
46
47 #saving the excel file and closing the csv file
48 wb.save(excel_name)
49 file.close()
```

The script above is a part of the [Awesome Python Scripts GitHub repository](#).

## 8. Pattern match with regular expressions

Collecting data from unstructured sources can be a very tedious process. Similar to the filtering example above, Python allows for more detailed pattern matching using regular expressions. This is useful for categorizing textual information as part of a data-processing workflow or searching for specific keywords in user-submitted content. The built-in regular expression library is called

`re`, and once you get the hang of the regular expression syntax, you can automate almost any pattern-matching script.

For example, maybe you'd like to match any email addresses found in the text you're processing. You can use this script to do so:

```
python
1 import re
2 emailRegex = re.compile(r'''(
3     [a-zA-Z0-9._%+-]+      # username
4     @                      # @ symbol
5     [a-zA-Z0-9.-]+        # domain name
6     (\.[a-zA-Z]{2,4})     # dot-something
7 )''', re.VERBOSE)
8
9 # store matched addresses in an array called "matches"
10 matches = []
11 text = """
12 An example text containing an email address, such as user@example.com or something like hello@e
13 """
14
15 # search the text and append matched addresses to the "matches" array
16 for groups in emailRegex.findall(text):
17     matches.append(groups[0])
18
19 # matches => ['user@example.com', 'hello@example.com']
20 print(matches)
```

You can use this script if you need to match phone numbers in your text:

```
python
1 import re
2
3 text = """
4 Here is an example string containing various numbers, some
5 of which are not phone numbers.
6
7 Business Address
8 4553-A First Street
9 Washington, DC 20001
10
11 202-555-6473
12 301-555-8118
13 """
14
15 phoneRegex = re.compile(r'''(
16     (\d{3}|(\d{3}\d{3}))?      # area code
17     (\s|-|\.)?                # separator
18     (\d{3})                    # first 3 digits
19     (\s|-|\.)                 # separator
20     (\d{4})                    # last 4 digits
21     (\s*(ext|x|ext.)\s*(\d{2,5}))? # extension
22 )''', re.VERBOSE)
23
24 matches = []
25 for numbers in phoneRegex.findall(text):
26     matches.append(numbers[0])
27
28 # matches => ['202-555-6473', '301-555-8118']
29 print(matches)
```

[Automate the Boring Stuff](#) has a [great chapter](#) about setting up and using regular expressions in Python.

## 9. Convert images to JPG

The `.jpg` format is perhaps the most popular image format currently in use. You may find yourself needing to convert images from other formats to generate project assets or image recognition. The `pillow` package from Python makes converting images to `.jpg` a simple process:

```
python
1 # requires the Pillow module used as `PIL` below
2 from PIL import Image
3 import os
4 import sys
5 file="toJPG.png"
6 filename = file.split(".")
7 img = Image.open(file)
8 new_name = filename[0] + ".jpg"
9 converted_img = img.convert('RGB')
10 converted_img.save(new_name)
```

## 10. Compress images

Sometimes you may need to compress an image as part of the asset-creation pipeline for a new site or temporary landing page and may not want to do so manually, or you have to send the task to an external image-processing service. Using the `pillow` package, you can easily compress JPG images to reduce the file size while retaining image quality. Install `pillow` using `pip install pillow`.

The example below will reduce a 2.5 MB image to 293 KB:

```
python
1 # the pillow package can be imported as PIL
2 from PIL import Image
3 file_path = "image_uncompressed.jpg"
4 img = Image.open(file_path)
5 height, width = img.size
6 compressed = img.resize((height, width), Image.ANTIALIAS)
7 compressed.save("image_compressed.jpg", optimize=True, quality=9)
```

## 11. Get content from Wikipedia

Wikipedia provides a fantastic general overview of many topics. This information can be used to add additional information to transactional emails, track changes on a particular set of articles, or to make training documentation or reports. Thankfully, it's also extremely easy to gather information using the Wikipedia package for Python.

You can install the Wikipedia package using `pip install wikipedia`. When the installation is complete, you are ready to begin.

If you already know the specific page content you would like to pull, you can do so directly from that page:

```
python
1 import wikipedia
2 page_content = wikipedia.page("parsec").content
3 # outputs the text content of the "Parsec" page on wikipedia
4 print(page_content)
```

This package also allows you to search for pages matching specified text:

```
python
1 import wikipedia
2 search_results = wikipedia.search("arc second")
3 # outputs an array of pages matching the search term
4 print(search_results)
```

## 12. Create and manage Heroku apps

Heroku is a popular platform for deploying and hosting web applications. As a managed service, it allows developers to easily set up, configure, maintain, and even delete applications through the [Heroku API](#). You can also easily create or manage Heroku applications using [Airplane runbooks](#) since Airplane makes it extremely easy to hit APIs and trigger events.

The example below relies on the `heroku3` package, which you can install using `pip install heroku3`. Note that you will need a [Heroku API key](#) to access the platform.

Connect to Heroku using Python:

```
python
1 import heroku3
2
3 # Be sure to update the api_key variable with your key
4 api_key = "12345-ABCDE-67890-FGHIJ"
5 client = heroku3.from_key(api_key)
```

Once you are connected to Heroku, you can list your available applications and select an application to manage directly:

```
python
1 import heroku3
2 api_key = "12345-ABCDE-67890-FGHIJ"
3 client = heroku3.from_key(api_key)
4
5 client.apps()
6
7 # the above command prints an array of available applications
8 # [<app 'airplanedev-heroku-example - ed544e41-601d-4d1b-a327-9a1945b743cb'>, <app 'notes-app -
9
10 # use the following command to connect to a specific application
11 app = client.apps()["airplanedev-heroku-example"]
12
13 # add a config variable for your application
14 config = app.config()
15 config["test_var"] = "value"
```

```
16
17 # enable or disable maintenance mode
18 # enable
19 app.enable_maintenance_mode()
20
21 # disable
22 app.disable_maintenance_mode()
23
24 # restarting your application is simple
25 app.restart()
```

Then, the following script will allow you to create an application as a part of an Airplane runbook:

```
python
1 import heroku3
2 api_key = "12345-ABCDE-67890-FABCD"
3 client = heroku3.from_key(api_key)
4
5 client.create_app("app-created-with-airplane")
```

After creating the application, you can manage it directly with the `heroku3` package. Head to [the Heroku3.py GitHub repository](#) for a full list of options.

## Get started

In this article we outlined twelve simple Python scripts you can use to automate various manual tasks. We selected these scripts not only because of their simplicity and utility but also because of their impact compared to their relative size.

Best of all, you can leverage [Airplane](#) to manage and run these scripts seamlessly and safely. Airplane runbooks allow you to easily manage and deploy Python scripts and compose multi-step workflows. Airplane also lets you manage your scripts on your local machine, which allows you to use your dev environment to write and test scripts before deployment.

While we covered Python in this article, you can also build workflows in Airplane using [Node.js or Docker](#) to run shell scripts. Airplane also allows you to work with SQL and REST, and automates alerting via [Slack](#) and email. You can [sign up for a free account](#) to try it out.

---

**Author:** [Alvin Charity](#)

airplane

*Alvin Charity is a writer, musician, sound artist, audio / video editor, and self-taught Javascript programmer based in Washington, DC.*



Share this article:



Subscribe to new blog posts from Airplane.