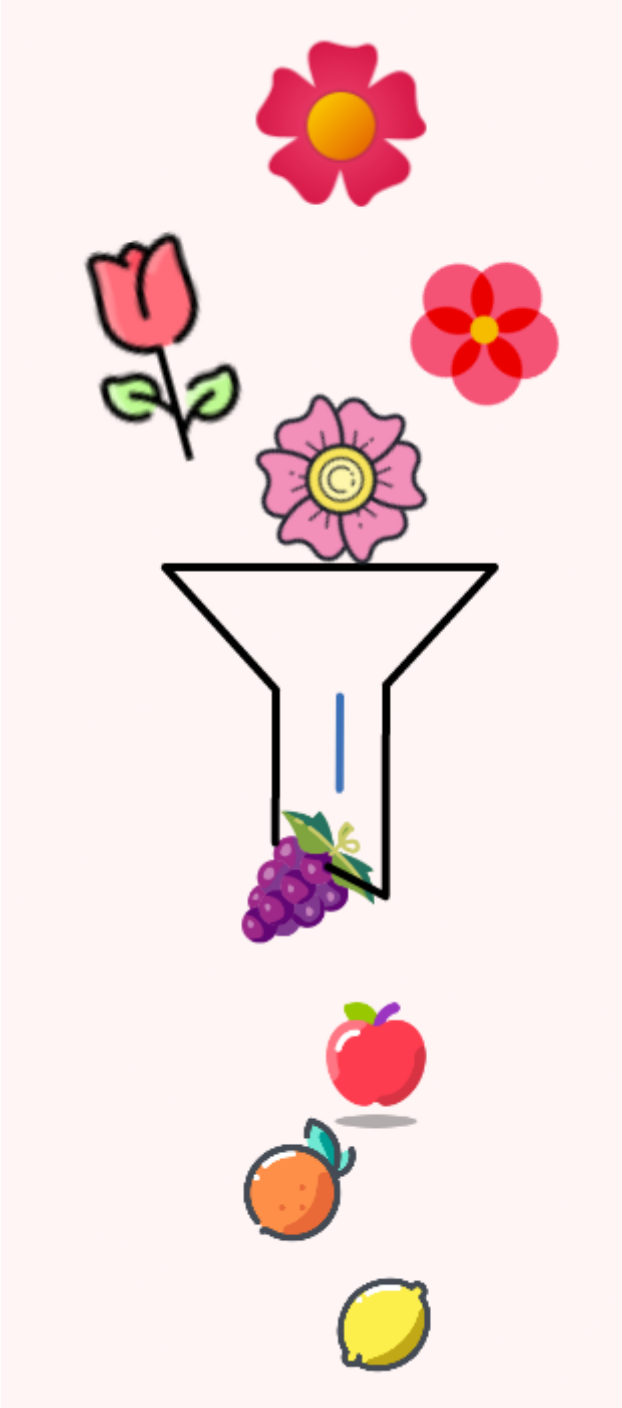
GitHub View on GitHub Book View Book

Best Practices for Writing Python Functions



Motivation

Have you ever looked at a function you wrote one month earlier and found it difficult to understand in 3 minutes? If that is the case, it is time to refactor your code. If it takes you more than 3 minutes to understand your code, imagine how long it would take for your teammates to understand your code.

If you want your code to be reusable, you want it to be readable. Writing clean code is especially important to data scientists who collaborate with other team members in different roles.

You want your Python function to:

- be small
- do one thing
- contain code with the same level of abstraction
- have fewer than 4 arguments
- have no duplication
- use descriptive names

These practices will make your functions more readable and easier to detect errors.

In this section, I will show you how to utilize the 6 practices mentioned above to write better Python functions.

Get Started

Let's start by taking a look at the function `load_data` below.

```
import xml.etree.ElementTree as ET
import zipfile
from os import listdir
from os.path import isfile, join

import gdown


def main():

    load_data(
        url="https://drive.google.com/uc?
id=1jI1cmxqnwsmC-vbl8dNY6b4aNBtBbKy3",
        output="Twitter.zip",
        path_train="Data/train/en",
        path_test="Data/test/en",
    )


def load_data(url: str, output: str, path_train: str,
path_test: str):

    # Download data from Google Drive
    output = "Twitter.zip"
    gdown.download(url, output, quiet=False)

    # Unzip data
    with zipfile.ZipFile(output, "r") as zip_ref:
        zip_ref.extractall(".")
```

```
    # Get train, test data files
    tweets_train_files = [
        file
        for file in listdir(path_train)
        if isfile(join(path_train, file)) and file !=
"truth.txt"
    ]
    tweets_test_files = [
        file
        for file in listdir(path_test)
        if isfile(join(path_test, file)) and file !=
"truth.txt"
    ]

    # Extract texts from each file
    t_train = []
    for file in tweets_train_files:
        train_doc_1 = [r.text for r in
ET.parse(join(path_train, file)).getroot()[0]]
        t_train.append(" ".join(t for t in
train_doc_1))

    t_test = []
    for file in tweets_test_files:
        test_doc_1 = [r.text for r in
ET.parse(join(path_test, file)).getroot()[0]]
        t_test.append(" ".join(t for t in
test_doc_1))

    return t_train, t_test


if __name__ == "__main__":
    main()
```

The function `load_data` tries to download data from Google Drive and extract the data. Even though there are many comments in this function, it is difficult to understand what this function does in 3 minutes. It is because:

- The function is awfully long
- The function tries to do multiple things
- The code within the function is at multiple levels of abstractions.
- The function has more than 3 arguments
- There are multiple duplications
- Function’s name is not descriptive

We will refactor this code by using the 6 practices mentioned above.

Small

A function should be small because it is easier to know what the function does. How small is small? There should rarely be more than 20 lines of code in one function. It can be as small as below. The indent level of a function should not be greater than one or two.

```
import zipfile

def unzip_data(output: str):

    with zipfile.ZipFile(output, 'r') as zip_ref:
        zip_ref.extractall('.')
```

Do One Task

A function should complete only one task, not multiple tasks. The function load_data tries to do multiple tasks such as download the data, unzip the data, get names of files that contain train and test data, and extract texts from each file.

Thus, it should be split into multiple functions like below

```
download_zip_data_from_google_drive(url, output_path)

unzip_data(output_path)

tweet_train, tweet_test =
get_train_test_docs(path_train, path_test)
```

And each function should do only one thing:

```
import gdown

def download_zip_data_from_google_drive(url: str,
output_path: str):

    gdown.download(url, output_path, quiet=False)
```

The function `download_zip_data_from_google_drive` only downloads a zip file from Google Drive and does nothing else.

One Level of Abstraction

The code within the function extract_texts_from_multiple_files is at a different level of abstraction from the function.

```
from typing import List

def extract_texts_from_multiple_files(path_to_file:
str, files: list) -> List[str]:

    all_docs = []
    for file in files:
        list_of_text_in_one_file =[r.text for r in
ET.parse(join(path_to_file, file_name)).getroot()[0]]
        text_in_one_file_as_string = ' '.join(t for t
in list_of_text_in_one_file)
        all_docs.append(text_in_one_file_as_string)

    return all_docs
```

> The level of abstraction is the amount of complexity by which a system is viewed or programmed. The higher the level, the less detail. The lower the level, the more detail. — PCMag

That is why:

- The function `extract_texts_from_multiple_files` is at a high-level of abstraction.
- The code

  `list_of_text_in_one_file =[r.text for r in ET.parse(join(path_to_file, file_name)).getroot()[0]]`

is at a low-level of abstraction.

To make the code within the function to be at the same level of abstraction, we can put the low-level code into another function.

```
from typing import List

def extract_texts_from_multiple_files(
    path_to_file: str, files: list
) -> List[str]:

    all_docs = []
    for file in files:
        text_in_one_file =
extract_texts_from_each_file(path_to_file, file)
        all_docs.append(text_in_one_file)

    return all_docs
```

```
def extract_texts_from_each_file(
    path_to_file: str, file_name: list
) -> str:

    list_of_text_in_one_file =[r.text for r in
ET.parse(join(path_to_file, file_name)).getroot()[0]]
    text_in_one_file_as_string = ' '.join(t for t in
list_of_text_in_one_file)

    return text_in_one_file_as_string
```

Now, the code extract_texts_from_each_file(path_to_file, file) is at a high-level of abstraction, which is the same level of abstraction as the function `extract_texts_from_multiple_files`.

Duplication

There is duplication in the code below. The part of code that is used to get the training data is very similar to the part of code that is used to get the test data.

```
t_train = []
for file in tweets_train_files:
    train_doc_1 =[r.text for r in
ET.parse(join(path_train, file)).getroot()[0]]
    t_train.append(' '.join(t for t in train_doc_1))

t_test = []
for file in tweets_test_files:
    test_doc_1 =[r.text for r in
ET.parse(join(path_test, file)).getroot()[0]]
    t_test.append(' '.join(t for t in test_doc_1))
```

We should avoid duplication because:

- It is redundant
- If we make a change to one piece of code, we need to remember to make the same change to another piece of code. If we forget to do so, we will introduce bugs into our code.

We can eliminate duplication by putting the duplicated code into a function.

```
from typing import Tuple, List

def get_train_test_docs(path_train: str, path_test:
str) -> Tuple[list, list]:
    tweets_train_files = get_files(path_train)
    tweets_test_files = get_files(path_test)

    t_train =
extract_texts_from_multiple_files(path_train,
tweets_train_files)
    t_test  =
extract_texts_from_multiple_files(path_test,
tweets_test_files)
    return t_train, t_test

def extract_texts_from_multiple_files(path_to_file:
str, files: list) -> List[str]:

    all_docs = []
    for file in files:
        text_in_one_file =
extract_texts_from_each_file(path_to_file, file)
        all_docs.append(text_in_one_file)

    return all_docs
```

Since the code to extract texts from training files and the code to extract texts from test files are similar, we put the repeated code into the function `extract_tests_from_multiple_files`. This function can extract texts from either training or test files.

Descriptive Names

> A long descriptive name is better than a short enigmatic name. A long descriptive name is better than a long descriptive comment. — Clean Code by Robert C. Martin

Users can understand what the function extract_texts_from_multiple_files does by looking at its name.

Don’t be afraid to write long names. **It is better to write long names rather than write vague names.** If you try to shorten your code by writing something like `get_texts` , it would be difficult for others to understand exactly what this function does without looking at the source code.

If the descriptive name of a function is too long such as `download_file_from_Google_drive_and_extract_text_from_that_file` . It is a good sign that your function is doing multiple things and you should split it into smaller functions.

Have Fewer than 4 Arguments

A function should not have more than 3 arguments since it is a sign that the function is performing multiple tasks. It is also difficult to test a function with more than 3 different combinations of variables.

For example, the function `load_data` has 4 arguments: `url`, `output_path`, `path_train`, and`path_test` . So we might guess that it tries to do multiple things at once:

- Use `url` to download data
- Save it at `output_path`
- Extract the train and test files in `output_path` and save it to `path_train` , `path_test`

If a function has more than 3 arguments, consider turning it into a class.

For example, we could split `load_data` into 3 different functions:

```
download_zip_data_from_google_drive(url, output_path)

unzip_data(output_path)

tweet_train, tweet_test =
get_train_test_docs(path_train, path_test)
```

Since the functions `download_zip_data_from_google_drive` , `unzip_data` , and `get_train_test_docs` are all trying to achieve one goal: get data, we could put them into one class called `DataGetter` .

```
import xml.etree.ElementTree as ET
import zipfile
from os import listdir
from os.path import isfile, join
from typing import List, Tuple

import gdown


def main():

    url = "https://drive.google.com/uc?
id=1jI1cmxqnwsmC-vbl8dNY6b4aNBtBbKy3"
    output_path = "Twitter.zip"
    path_train = "Data/train/en"
    path_test = "Data/test/en"

    data_getter = DataGetter(url, output_path,
path_train, path_test)

    tweet_train, tweet_test =
data_getter.get_train_test_docs()


class DataGetter:
    def __init__(self, url: str, output_path: str,
path_train: str, path_test: str):
        self.url = url
        self.output_path = output_path
        self.path_train = path_train
        self.path_test = path_test
        self.download_zip_data_from_google_drive()
```

```
        self.unzip_data()

    def download_zip_data_from_google_drive(self):

        gdown.download(self.url, self.output_path,
quiet=False)

    def unzip_data(self):

        with zipfile.ZipFile(self.output_path, "r")
as zip_ref:
            zip_ref.extractall(".")

    def get_train_test_docs(self) -> Tuple[list,
list]:

        tweets_train_files =
self.get_files(self.path_train)
        tweets_test_files =
self.get_files(self.path_test)

        t_train =
self.extract_texts_from_multiple_files(
            self.path_train, tweets_train_files
        )
        t_test =
self.extract_texts_from_multiple_files(
            self.path_test, tweets_test_files
        )
        return t_train, t_test

    @staticmethod
    def get_files(path: str) -> List[str]:

        return [
            file
            for file in listdir(path)
```

```
            if isfile(join(path, file)) and file !=
"truth.txt"
        ]

    def extract_texts_from_multiple_files(
        self, path_to_file: str, files: list
    ) -> List[str]:

        all_docs = []
        for file in files:
            text_in_one_file =
self.extract_texts_from_each_file(path_to_file, file)
            all_docs.append(text_in_one_file)

        return all_docs

    @staticmethod
    def extract_texts_from_each_file(path_to_file:
str, file_name: list) -> str:

        list_of_text_in_one_file = [
            r.text for r in
ET.parse(join(path_to_file, file_name)).getroot()[0]
        ]
        text_in_one_file_as_string = " ".join(t for t
in list_of_text_in_one_file)

        return text_in_one_file_as_string


if __name__ == "__main__":
    main()
```

In the code above, I use `staticmethod` as the decorators for some methods because these methods do not use any class attributes or class methods. Find more about these methods here.

As we can see, none of the functions above have more than 3 arguments! Even though the code that uses a class is longer compared to the code that uses a function, it is much more readable! We also know exactly what each piece of code does.

How do I write a function like this?

Don’t try to be perfect when starting to write code. Start with writing down complicated code that matches your thoughts. Then as your code grows, ask yourself whether your function violates any of the practices mentioned above. If yes, refactor it. Test it. Then move on to the next function.