A beginner's guide on how to build a simple Python application with a command line interface

Anna Cole · Follow
7 min read · Feb 1, 2024

If you are facing the challenge to build your first project in Python and don't know how to start, you are not alone. Learning a new technology and not knowing how to proceed can be an intimidating experience. However, having a step-by-step guide for tackling the task can help you to gain confidence, organize your thought process, and get your own project off the ground. With practice, creating an application can become a much easier and enjoyable experience. This article is a beginner's guide where I teach you my approach on how to write a simple Python program with a command line interface. We will build a program that keeps track of books of a library, but you can adapt it to your needs, changing names and data accordingly. In the end of this tutorial I will also guide you on how to connect the app to a Github repository, so you can save and share your code.

Step 1 — Open your terminal and create a folder for your project.

For this project, you will have to install Python in your computer. Assuming you have already done it, open your terminal and navigate to a location where you would like your application directory to be located. Then, create a name for you app. In my example I named it "library". In the terminal, run the following command, with your app's name instead of "library":

```
mkdir library
```

Now, let's navigate into it by changing directories, so run this command in your terminal (remember to type your app's name instead of "library"):

```
cd library
```

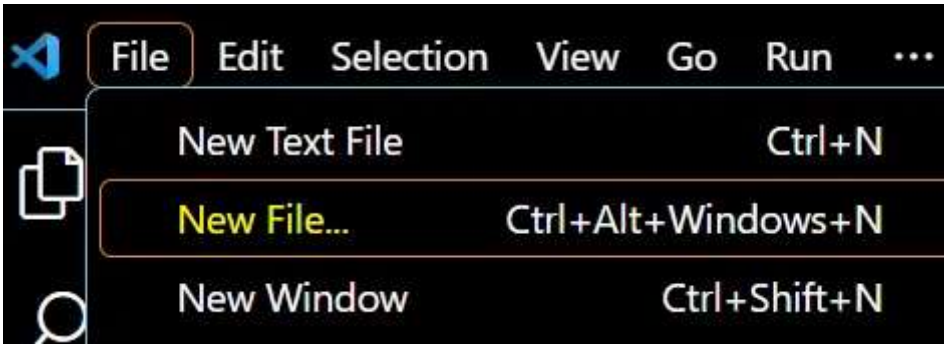
Step 2 — Start coding your application

To start coding your app, you need to open it in a code editor program. I use VS Code for that, as, in my opinion, it is very user-friendly.

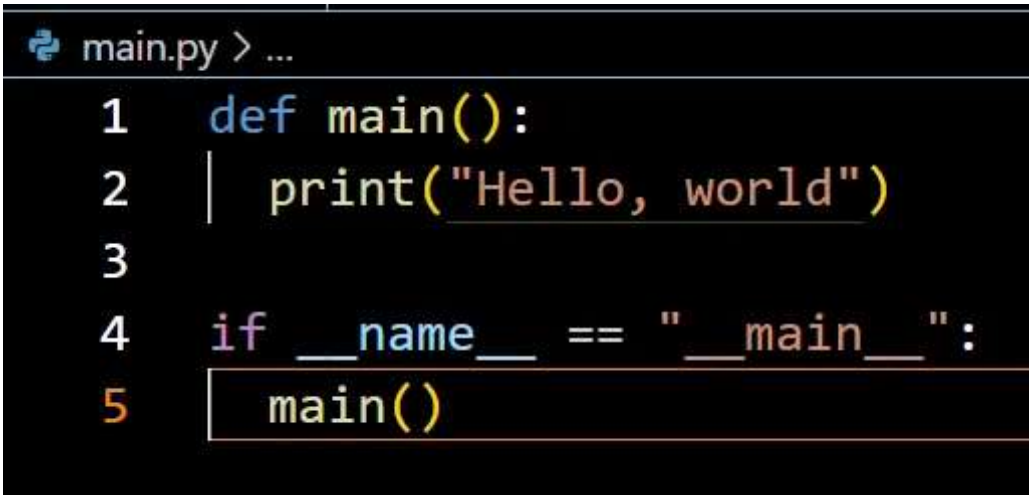
So, assuming you already have VS Code installed, type the following in your terminal. Make sure you are in your project's directory before running the command.

```
code .
```

This command will open your project on VS Code. Go to the top left corner and create a new file with a python extension. I called mine "main.py". This will be our main file, where we are going to type our code.



To test it out, let's create a main function in main.py file, and inside it we will print something to check if the function is working.



Let’s execute the file to test the function. For that, open the integrated terminal in VS Code by pressing the ctrl and ` keys at the same time. Then, type “python main.py” in the terminal and you should see “Hello, world” printed in the terminal.

Delete the print statement from the function to start coding your app. When you are testing your code, remember to always exit the program and run it again, if you change anything in your code. This way the new changes will take effect.

P.S. The code on lines 4 and 5 automatically executes the “main” function, after you run “python main.py” in the terminal.

Step 3— Create a CLI menu.

Let’s think about the interface of this program. I want this program to have a menu driven interface, in which the user will be prompted to choose an option and the program will perform an action based on that option.

Having said that, we will need to create loops in which the user will navigate, until he chooses to exit the loop and terminate the program.

We can have many choices and loops as we want, but for demonstration we will only have 4 choices in our example.

Let’s print the choices:

```python
def main():
    choice = 0
    while choice != 4:
        print("Library Manager")
        print("1. Add a book")
        print("2. Find a book")
        print("3. Show all books")
        print("4. Exit")
        choice = int(input())

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

As long as the choice meets the condition (choice is not 4) then the user will be in the loop. However, if the choice is 4, then the loop will be terminated.

Step 4 — Create features and logic

We can now start building the logic for the functionalities we want. So, let’s write some conditional statements for the difference choices.

```python
if choice == 1:
    print("Adding a book")
    title = input("Enter the book's title: ")
    gender = input("Enter the book's gender: ")
    author = input("Enter the book's author: ")
```

This conditional statement will prompt the user to type the title, gender and author of a new book, if the choice is 1.

But where this input data will go? It must be stored somewhere. Normally, developers build a database that will store the data. However, for the sake of simplicity we will store the data in a list.

So, let’s initialize an empty list that will store the given data. Then, we write the logic to append this data to the books list.

```
def main():

    books = []
    choice = 0
    while choice != 4:
        print("Library Manager")
        print("1. Add a book")
        print("2. Find a book")
        print("3. Show all books")
        print("4. Exit")
        choice = int(input())

        if choice == 1:
            print("Adding a book")
            title = input("Enter the book's title: ")
            gender = input("Enter the book's gender: ")
            author = input("Enter the book's author: ")
            books.append([title, gender, author])
```

The next choice will be to find a certain book. For that, we will iterate through the books list to search for this particular book.

```
elif choice == 2:
    print("Finding a book")
    search_word = input("Enter search word: ")
    for book in books:
        if search_word in book:
            print(book)
```

Now, let’s write the logic for the choice that will display all the books stored in the books list:

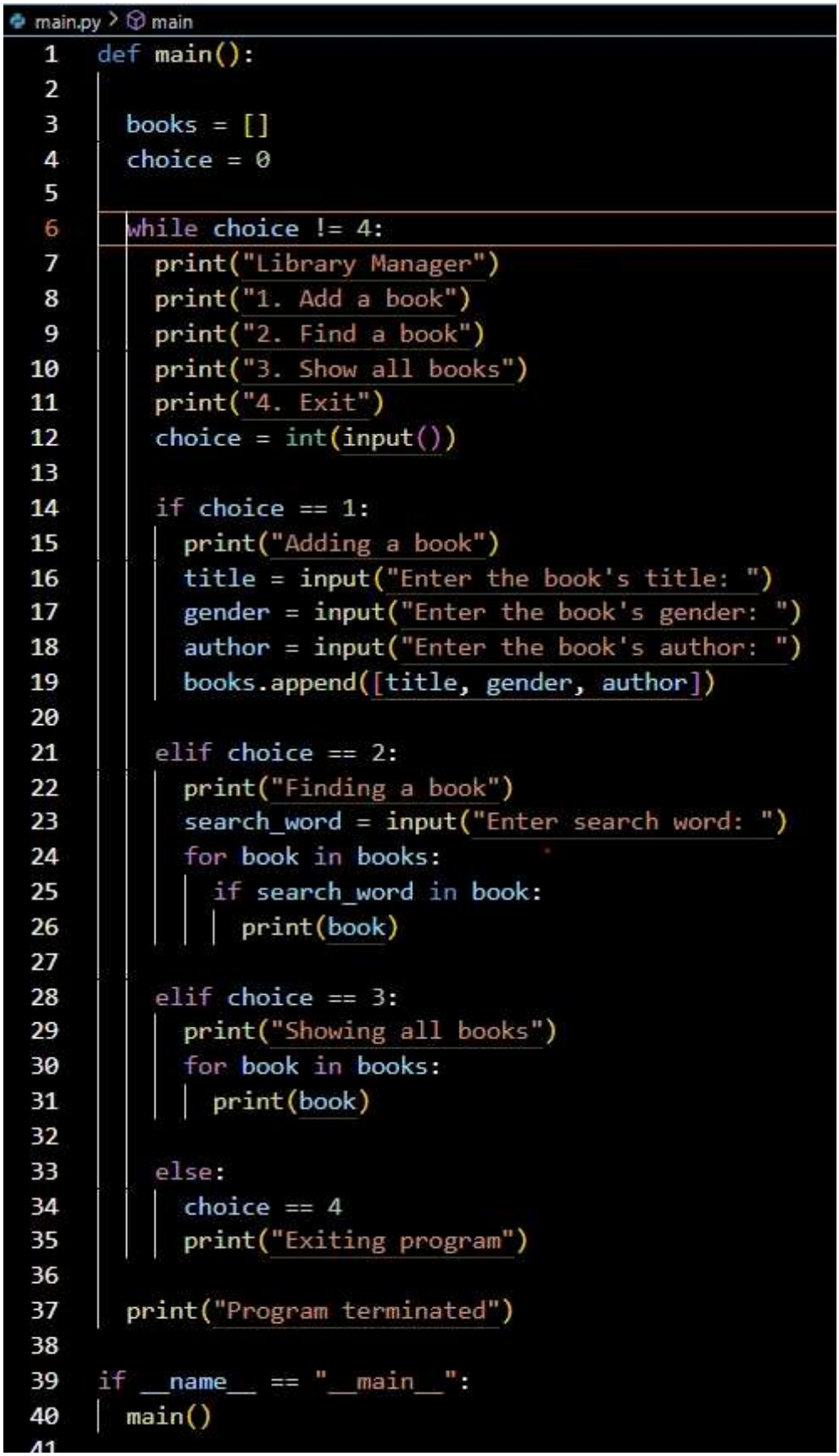
```
elif choice == 3:
    print("Showing all books")
    for book in books:
        print(book)
```

Finally, let’s print a message to inform the user that they are exiting the program, if choice is number 4. Outside the loop we will print a message to inform that the program was terminated.

```
    else:
        choice == 4
        print("Exiting program")

print("Program terminated")
```

The whole code will look like this:



```python
def main():

    books = []
    choice = 0

    while choice != 4:
        print("Library Manager")
        print("1. Add a book")
        print("2. Find a book")
        print("3. Show all books")
        print("4. Exit")
        choice = int(input())

        if choice == 1:
            print("Adding a book")
            title = input("Enter the book's title: ")
            gender = input("Enter the book's gender: ")
            author = input("Enter the book's author: ")
            books.append([title, gender, author])

        elif choice == 2:
            print("Finding a book")
            search_word = input("Enter search word: ")
            for book in books:
                if search_word in book:
                    print(book)

        elif choice == 3:
            print("Showing all books")
            for book in books:
                print(book)

        else:
            choice == 4
            print("Exiting program")

    print("Program terminated")

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

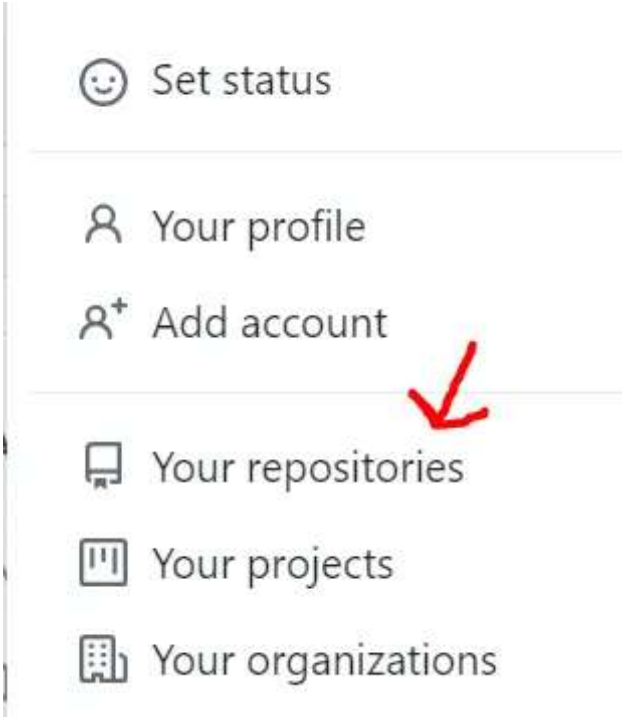
To test the application, add some books and then choose some options to see if the function is working, like in the example below:

That’s it! We have just created a simple Python app with a command line interface that can store and manipulate data. Obviously, in order to store and persist data in a real production environment, you would have to create database tables and additional functions to manipulate data to and from those tables.

Before finishing this tutorial, there is just one last step I want to show you, so you can save your code and share it with other people.

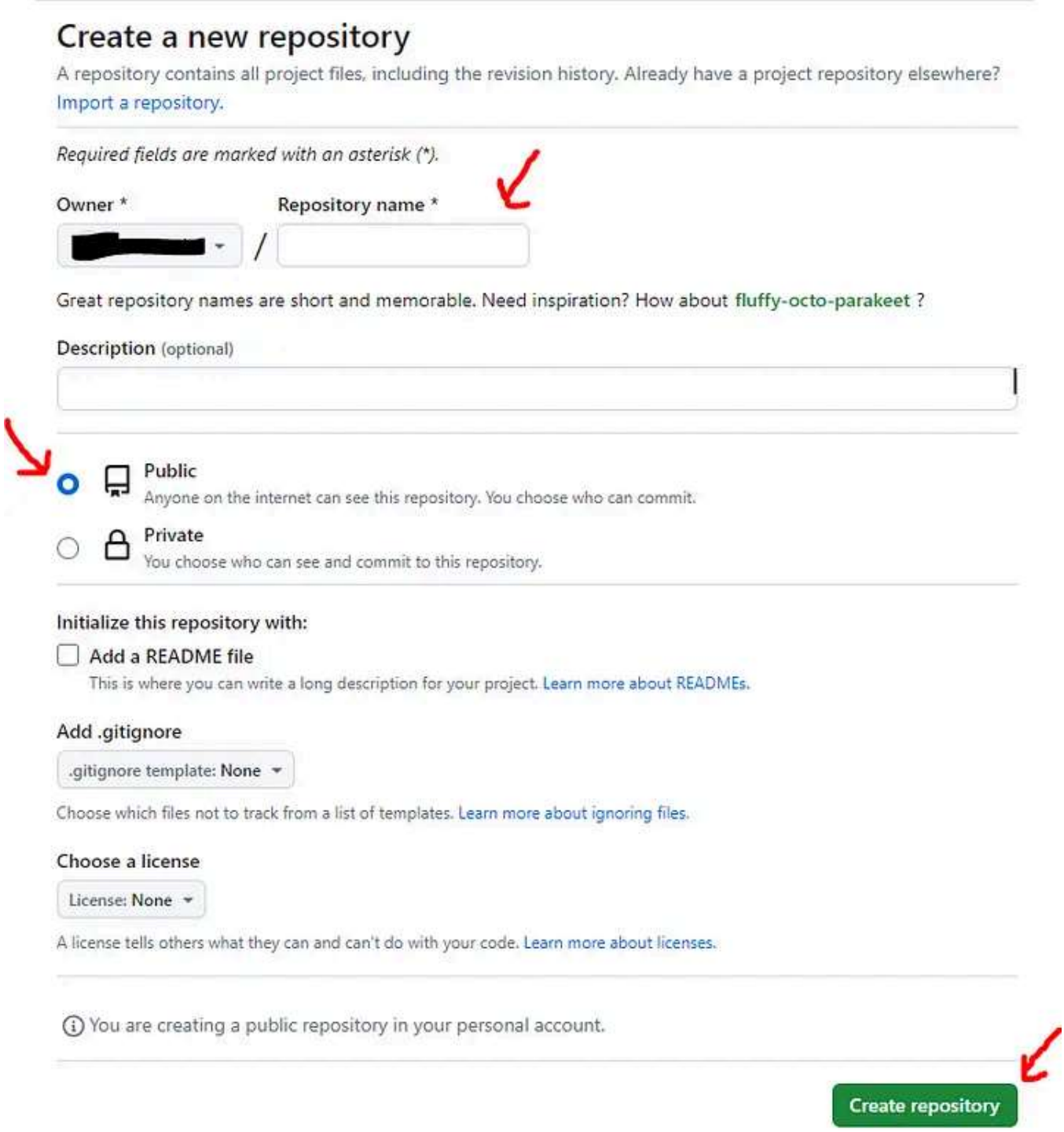
Step 5— Connect your app to a Github repository

Assuming you already have an account at Github, click on your profile picture, then go to “your repositories”.



Click on the “new” green button at the top-right corner, to create a new repository.

Look at the field “Owner” and make sure you are the owner of the repo. Then, type your repository name, which should match the name of your app. Select the “public” option and click “create repository”, like in the snippet below:



Now that you have created your remote repo, you need to connect it with your app, so you can start committing and pushing changes.

Go back to your terminal, make sure you are in your app's folder and type "git init". This will initialize your repo in the terminal.

```
$ git init
```

Then, run "git remote add origin" followed by the URL of the remote repo, like in the example below (copy the repo's URL from the browser's address bar).

```
$ git remote add origin https://github.com/barbie/your-app-name
```

After that, run “git add .”, then run “git commit -m ‘initial commit’”.

```
$ git add .
```

```
$ git commit -m 'initial commit'
```

Then, run “git push -u origin”, to have your changes saved in the remote repository.

```
$ git push -u origin.
```

Conclusion

Congratulations, you now have created your first Python command line interface app and connected it to a Github repo! You can now adapt the app to your needs, substituting books for other data of your choice and adding more functionalities.

Thank you for reading. Please leave a like if you think this article is useful, and leave your comment or feedback. Also, if you are a beginner, like me, always remember that one small step at a time will get you there! Happy coding! :)

Sources:

Github

Python | Command Line Interface | Begginer | Python Programming | Command Line