Python Learning Journal

Guy Rimel

About Myself

I'm just a normal Guy. I'm 32 years old, I live in Texas, I love Jesus, and my wife, and good coffee. I took a coding class in college (back in 2017) and it has piqued my interest ever since. Though, only in the last year have I gotten serious about software.

I enjoy the basic full-stack, and I'm a web-content creator for Texas Tech University, but making websites is simple stuff. Diving into packages, frameworks, backends, APIs, and more complex languages is equal parts fun and headache inducing.

1.1: Getting Started With Python

Learning Goals

- Summarize the uses and benefits of Python for web development
- Prepare your developer environment for programming with Python

Okay, starting off my Python journal here. Day one saw me installing Python with the Windows x64 installer. I'm running out of disk space on my laptop. The installer does it's thing, I open a command prompt and initiate python's built in REPL (Read Execute Print Loop) (is that pronounced "repple"?), then type in ```print("Hello World!")``` and just like that, I've mastered Python!

I learned about "environments" which are directories of particular Python packages and files to separate dependencies.

Another critical thing I learned was the *import* keyword, to utilize Python's built-in package management system, e.g., ```import math``` will bring in a package with mathematical functionality.

Similar to *import* is the keyword *pip* which installs packages from the Python Package Inventory. So, ```pip intall xyz``` will install the xyz package in the current directory location from the terminal.

Python is dynamically typed, meaning that a variable can be assigned a string value, then reassigned as an integer value, with no qualms (like JavaScript).

Python lines do not have semicolons…

1.2: Data Types in Python

Learning Goals

- Explain variables and data types in Python
- Summarize the use of objects in Python
- Create a data structure for your Recipe app

Python data types are more fun than riding a jet ski. We all know that Python has data types of string, integer, boolean, tuple, list, dictionary, and noneType (and there are probably others).

It was good to practice the syntax of built-in functions and methods on different data types. How to “slice” properly, sort(), min(), max(), count(), pop(), extend(), append(), copy(), and more.

Apparently, objects in Python are called “dictionaries” allowing for an entity of key/value pairs of different data types.

For the basic data structures of the Achievement’s recipe app, I’ve chosen to store all recipes as a list of dictionaries (each recipe is a “dictionary”). This allows for standardized key/value pairs for all recipes.

1.3: Operators & Functions in Python

Learning Goals

- Implement conditional statements in Python to determine program flow
- Use loops to reduce time and effort in Python programming
- Write functions to organize Python code

Conditional statements execute code based on a boolean value. That’s it, that’s what they do: they execute a code block based on “True” or “False”. Start a conditional statement with the “if” keyword, then a value, then desired comparators, if any, like “AND” or “OR” or “NOT” and another value. End it with a colon. Indent. Now, write a code block to be executed if the statement is truthy. Now, add “elif” or “else” or perhaps *nest* another “if”.

The two loops discussed in this chapter were for-loops and while-loops. The for-loop will execute a code block for a certain number of times as specified by an “iterable”, which would be

anything with a “length” (array, tuple, string). The while-loop will run the code block FOREVER… until the conditional statement becomes falsy. If you said ```while True: ``` the code block would be executed in an infinite loop, and everyone would laugh and point at how terrible it is, forever.

In a loop, use “break” or “continue” to either exit the loop, or skip an iteration.

Functions are reusable code blocks that can be passed arguments to do or return something. Start a Python function with “def” to define the function, then the function name, then two parentheses “()” and if desired, throw some parameters in those parentheses. Parameters can be assigned default values with the equals sign.

1.4: File Handling in Python

Learning Goals

- Use files to store and retrieve data in Python

Using files to store and retrieve data in Python. This was a fresh new concept for me, but obviously a very important topic to create any meaningful Python-based software.

For basic (text) reading and writing, declare a file path and a method flag as arguments of the open function like so ``` file_variable = open(‘./thing.txt’, ‘rt’) ``` the t is for “text” and it’s implied by default. Then use the readlines() method to read the data: ``` print(file_variable.readlines()) ``` to return an array of each line in the text file, and finally, CLOSE the file with the .close() method.

To READ a text file, you would use something like: ``` file_variable = open(‘./thing.txt’, ‘w’) ``` then the method .writelines(array_name) would write each element of an iterable. But all the items would smoosh together, so be sure to include a newline character “\n” after each iteration.

Okay, now pickles. Use pickles to store more complex data types (dictionaries). First ``` import pickles ```, then, let’s say the dictionary is named “my_dictionary”, DUMP the dictionary into a .bin file like this: ```file_variable = open(‘thing.bin’, ‘wb’) ``` then ``` pickle.dump(my_dictionary, file_variable ``` then ``` file_variable.close() ```

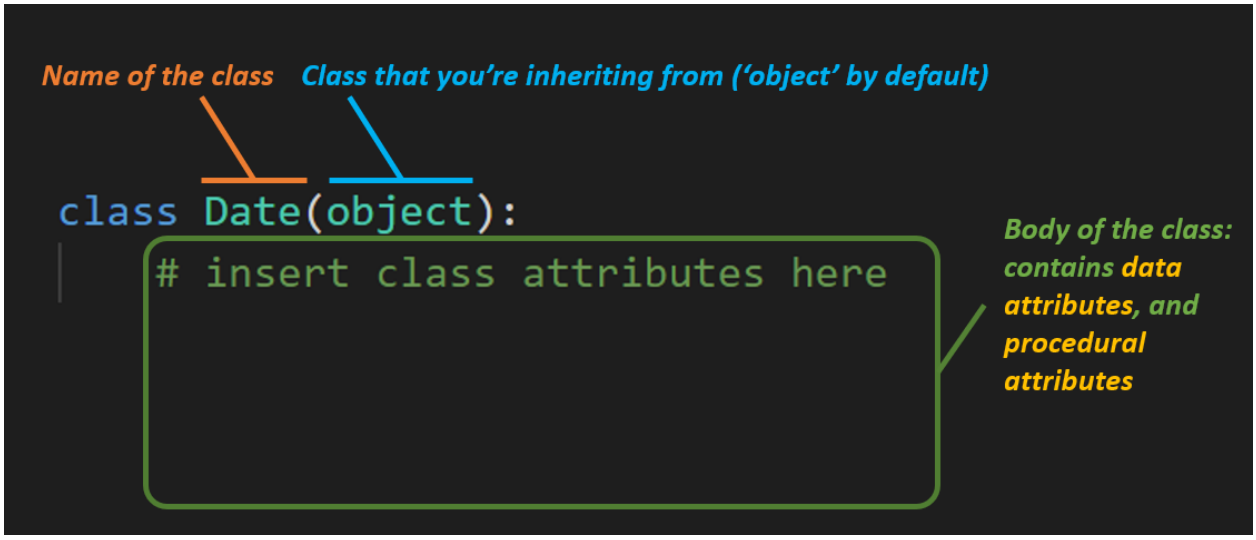
To read pickles, try the following: ``` with open(‘file_name.bin’, ‘rb’) as file_rb: \n data = pickle.load(file_rb) ```

1.5: Object-Oriented Programming in Python

Learning Goals

- Apply object-oriented programming concepts to your Recipe app

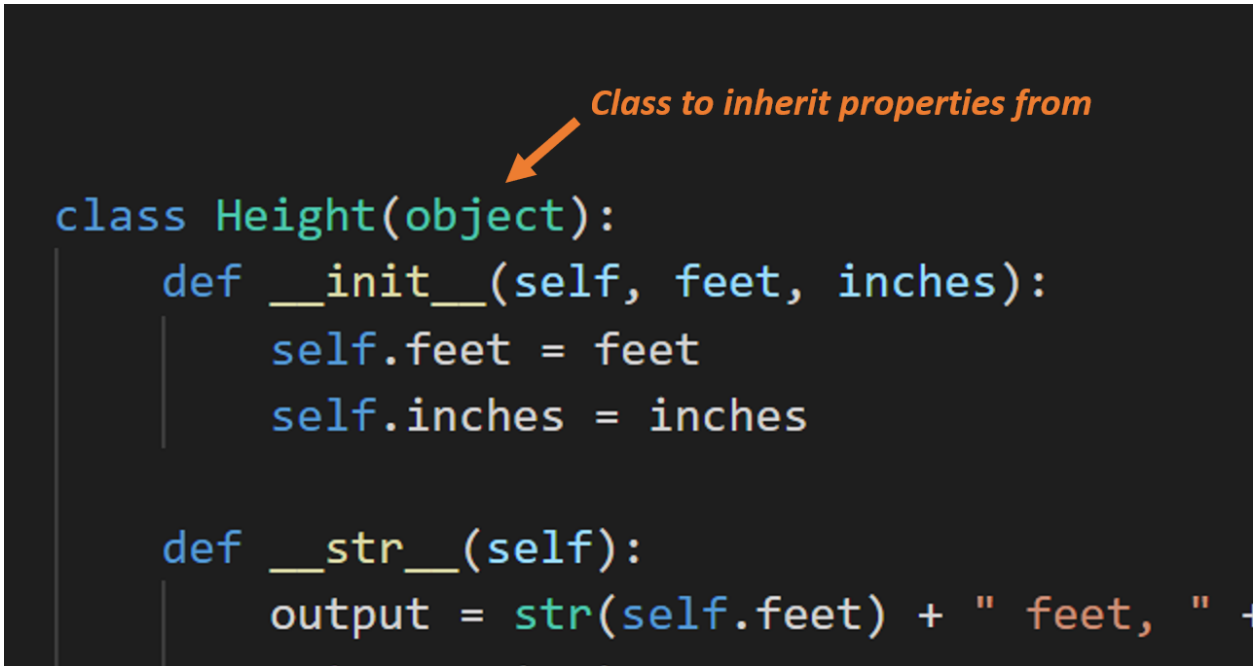
What is Object-Oriented Programming? OOP involves abstracting data into objects to keep the code “dry” (don’t repeat yourself). “Everything in Python is an object, based on a class.” A class is like a template of an object. It is comprised of “data attributes” and “procedural attributes”. Data attributes are just stored values, e.g., ‘name’: ‘Bob’. Procedural attributes are methods that perform preconfigured functions.



Defining data attributes:

```
class Date(object):
    def __init__(self, day, month, year):
        self.day = day
        self.month = month
        self.year = year
```

Inheritance:




```
class Person:
    def walk():
        print("Hello, I can walk!")
```

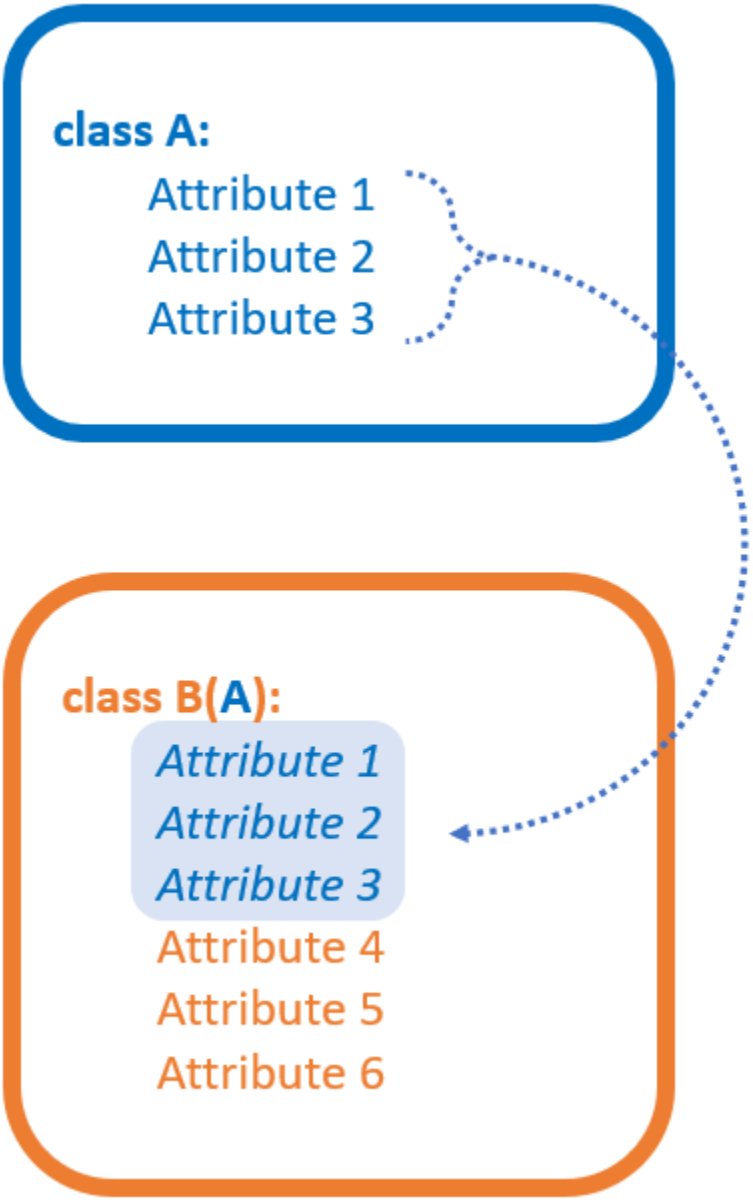
First, note how when building a class for the first time, you don't actually need to specify the `object` parameter yet (i.e., the class that's being inherited from). You can just enter `class` followed by the name of the class you're building (e.g., `class Height`).

Consider `Person` as your base class, and assume a new class called `Athlete`. You'll have `Athlete` inherit all of `Person`'s properties, while defining another method under it, called `run()`:

```
class Athlete(Person):
    def run():
        print("Hey, I can run too!")
```

The above text (directly from CareerFoundry) is important! This shows that you can define a class, then define another class taking the first class as an argument. Which will inherit all of the first class's attributes!

Like so:



1.6: Databases in Python

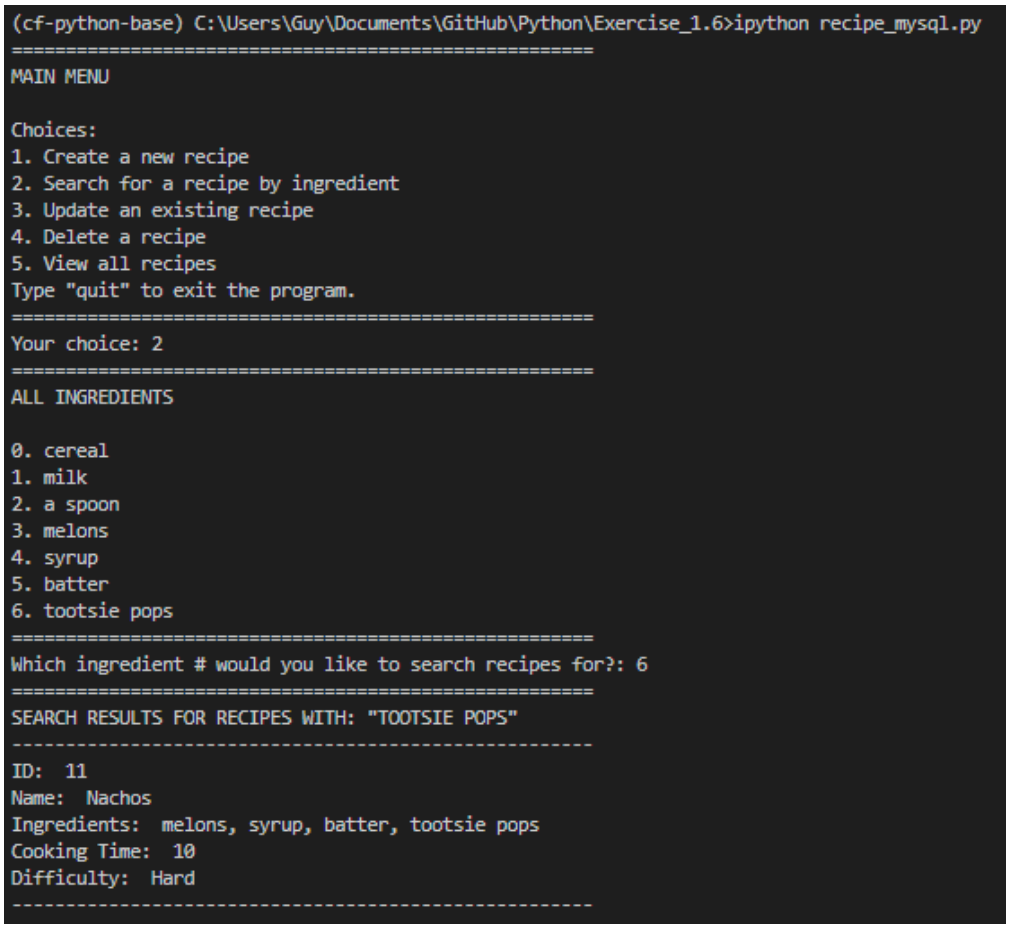
Learning Goals

- Create a MySQL database for your Recipe app

The MySQL database was a free download, with relatively easy setup. Though CareerFoundry introduced me to working with databases by diving into MongoDB (a non-relational database) I have started to warm up to SQL. The syntax is relatively natural, and the CRUD (Create = INSERT, Read = READ, Update = UPDATE, Delete = DELETE) operations are not brain-surgery to interact with.

I love spreadsheets, Excel is great, and SQL captures the charm of columns, rows, and cells when structuring data. Step aside, MongoDB. SQL is great.

Though, it is tricky to enter SQL syntax into Python… two languages at once.



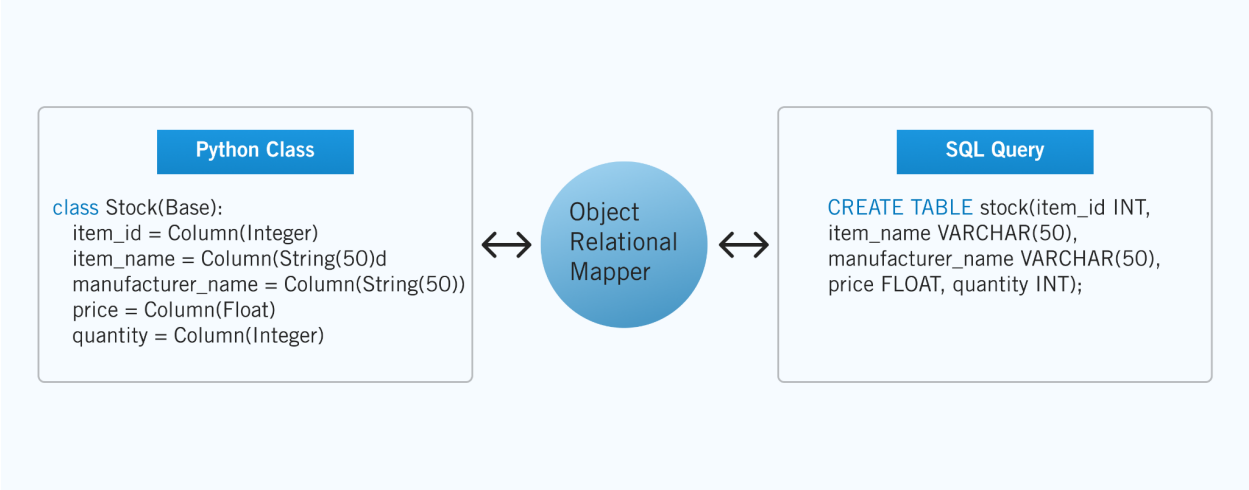
1.7: Object-Relational Mapping in Python

Learning Goals

- Interact with a database using an object-relational mapper
- Build your final command line Recipe application

Here's an example of an item being added to "Stock" with ORM:

```
new_item = Stock(
        item_id = 1,
        item_name = "Water",
        manufacturer_name = "Aquafina",
        price = 10,
        quantity = 20
)

session.add(new_item)
session.commit()
```



Object Relational Mapping allows for interactions with data from a SQL database to be treated as Python objects. This means that instead of using Python, AND SQL syntax, you can just use Python.

2.1: Getting Started with Django

Learning Goals

- Explain MVT architecture and compare it with MVC
- Summarize Django's benefits and drawbacks
- Install and get started with Django

A key concept to grasp is the definition of MVT and how it differs from MVC. MVC is "Model, View, Controller" while MVT is "Model View Template".

With MVC, Model is the Database. View is what renders and collects data in the browser. Controller is the logic between the model and the view.

With MVT, Model is the Database. View is the business logic that fetches from the database and gives to the frontend. Template is the user interface and renders things in the browser.

Pre-Requisite Checks

Python: version 3.8.7

```
(web-dev) C:\Users\Guy>python --version
Python 3.8.7
```

Activated Virtual Environment: named "web-dev"

```
(web-dev) C:\Users\Guy>C:\Users\Guy\Envs\web-dev\Scripts\activate.bat

(web-dev) C:\Users\Guy>if defined _OLD_VIRTUAL_PYTHONPATH (set "PYTHONPATH=" )  else (set "_OLD_VIRTUAL_PYTHONPATH=" )
```

Django installed in the environment:

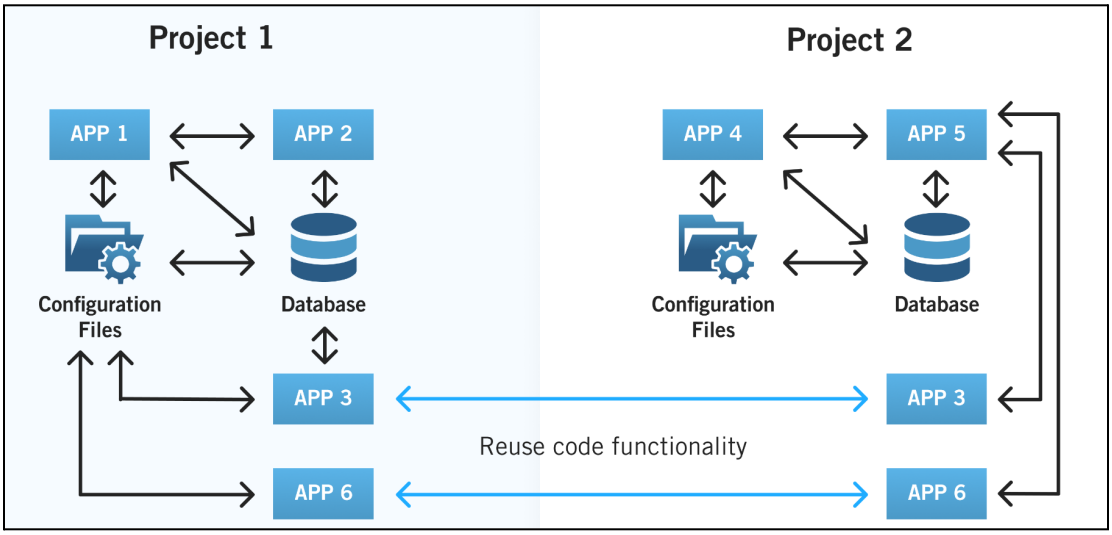
```
(web-dev) C:\Users\Guy>py -m pip install Django
Collecting Django
  Downloading Django-4.2.2-py3-none-any.whl (8.0 MB)
     ---------------------------------------- 8.0/8.0 MB 2.2 MB/s eta 0:00:00
Collecting asgiref<4,>=3.6.0 (from Django)
  Downloading asgiref-3.7.2-py3-none-any.whl (24 kB)
Collecting sqlparse>=0.3.1 (from Django)
  Downloading sqlparse-0.4.4-py3-none-any.whl (41 kB)
     ---------------------------------------- 41.2/41.2 kB 998.2 kB/s eta 0:00:00
Collecting backports.zoneinfo (from Django)
  Downloading backports.zoneinfo-0.2.1-cp38-cp38-win_amd64.whl (38 kB)
Collecting tzdata (from Django)
  Downloading tzdata-2023.3-py2.py3-none-any.whl (341 kB)
     ---------------------------------------- 341.8/341.8 kB 2.1 MB/s eta 0:00:00
Collecting typing-extensions>=4 (from asgiref<4,>=3.6.0->Django)
  Downloading typing_extensions-4.6.3-py3-none-any.whl (31 kB)
Installing collected packages: tzdata, typing-extensions, sqlparse, backports.zoneinfo, asgiref, Django
Successfully installed Django-4.2.2 asgiref-3.7.2 backports.zoneinfo-0.2.1 sqlparse-0.4.4 typing-extensions-4.6.3 tzdata
-2023.3

(web-dev) C:\Users\Guy>django-admin --version
4.2.2
```

2.2: Django Project Set Up

Learning Goals

- Describe the basic structure of a Django project
- Summarize the difference between Django projects and apps
- Create a Django project and run it locally
- Create a superuser in the admin interface of a Django web application

In Django a “Project” is the entire application structure. An “App” is a module with specific functionality. Apps are reusable pieces to keep things dry and use in multiple projects.



Creating a Django Project Step-by-Step

```
mkvirtualenv web-dev
```

```
web-dev\Scripts\activate.bat
```

```
pip install django
```

```
django-admin.exe startproject bookstore
```

Some possible commands that you can run via `manage.py` (or `django-admin`) are:

COMMAND	ACTION
`check`	Checks the framework for common problems
`migrate` or `makemigrations`	Creates/Updates database
`runserver`	Runs a test server
`diffsettings`	Checks for differences from default settings
`sendtestemail`	Sends a test email
`startapp`	Creates an app
`test`	Runs tests

Run Migrations:

```
...\> py manage.py migrate
```

Run Server:

```
py manage.py runserver
```

Create an App (“books” app):

```
...\> py manage.py startapp books
```

Create superuser:

```
python manage.py createsuperuser
```