Classes and Objects

Why is this important?

Classes and objects are a way to organize your code and data. They are a way to model real world objects in code. There's a whole bunch of benefits of using classes and objects, let's take a look at a few of them.

- Encapsulation
  - This is a fancy word that means that you can group data and functions together. This is a way to organize your code and data.
- Reusability/Code maintenaince
  - You can reuse classes and objects in your code. This is a way to organize your code and data.
- Abstraction
  - This is a fancy word that means that you can hide the details of how something works. This is a way to organize your code and data.
- Many other benefits that you'll learn more about the more you program like inheritance, polymorphism and much more.

When programming in Python and using the immense ecosystem, many times you'll inherit from classes that other people have written. This is an awesome way to supercharge your code and get a lot of functionality for free.

What are we going to do?

We're going to build an application that will help us keep track of books in a library. We'll be able to add books to our library, remove books from our library, and see all the books in our library.

In this example, we'll have two classes, a `Book` class and a `Library` class. The `Book` class will represent a single book and the `Library` class will represent a collection of books.

Steps

1. let's create a file named `library_app.py` in this folder. This is going to be where we write the main pieces of our code. Let's also add a bit of code here:

```python
if __name__ == '__main__':
    print("Welcome to our library App")
    print("---------------------------")
```

2. We're going to create a directory named `library_tools` inside of the current folder so that we can keep our code organized.

- Inside of the library folder to be able to import anything into this folder, we need to create a file named `__init__.py`. This file can be empty, but it needs to exist.
  - Essentially this file is telling Python that this folder is a Python package and that it can be imported.

3. Inside of the `library_tools` folder, let's create a file named `book.py`. This is where we'll write our `Book` class. Let's create it below.

```python
class Book:
    def __init__(self, title, author, pages):
        self.title = title
        self.author = author
        self.pages = pages
```

- this essentially will be our blueprint for what a book is, but this isn't a book yet. In the next step, we'll import this into our `library_app.py` file and create a book.
- Note that in Python classes are named in `PascalCase` and functions are named in `snake_case`. This is a convention that is used in Python.
- the function `__init__` is called a constructor. This is a special function that is called when you create an instance of a class.
    - The first argument of this function is always `self`. This is a reference to the instance of the class that is being created.
    - The other arguments are the arguments that are passed in when the class is created.
    - In this case, we're passing in `title`, `author`, and `pages`.
    - We're then setting the attributes of the class to the values that are passed in.

4. Let's import our `Book` class into our `library_app.py` file and create a book. We can do this by adding the following line to the top of our file.

```python
from library_tools.book import Book

if __name__ == '__main__':
    print("Welcome to our library App")
    print("--------------------------")
    book = Book("The Lord of the Rings", "J.R.R. Tolkien", 1000)
    # let's print the books properties.
    print("The properties of our book are:")
    print(f"Title: {book.title}")
    print(f"Author: {book.author}")
    # let's print the book.
    print("The book is:")
    print(book)
```

- now if you run the program you'll get output that looks like this.

```
$ python library_app.py
Welcome to our library App
--------------------------
The properties of our book are:
Title: The Lord of the Rings
Author: J.R.R. Tolkien
```

```
The book is:
<library_tools.book.Book object at 0x0000011EF5A57D60>
```

- Note that the last line here is a little weird. This because we haven't told python how to print our book. We're going to use *double underscore* also called *dunder* methods to do this.

5. Let's add a `__str__` method to our `Book` class. This is a special method that is called when you try to convert an object to a string. Let's add it to our `Book` class in the `library_tools/book.py` file.

```
class Book:
    def __init__(self, title, author, pages):
        self.title = title
        self.author = author
        self.pages = pages

    # representation of our object when
    # we print it out in a string.
    def __str__(self):
        return f"{self.title} by {self.author}"
```

- note that we're using the `self` keyword to access the attributes of the class.
- now if you run the program you'll get output that looks like this.

```
$ python library_app.py
Welcome to our library App
--------------------------
The properties of our book are:
Title: The Lord of the Rings
Author: J.R.R. Tolkien
The book is:
The Lord of the Rings by J.R.R. Tolkien
```

- Now if you look at the last line you have a nice representation of the book. This is much better than the output we had before.
- `__init__` and `__str__` are just two of the many dunder methods that you can use in Python. You can find a list of them here. Here's a couple of examples that you could do with other dunder methods:
  - if you wanted to compare two books to see if they were the same, you could use the `__eq__` method.
  - if you wanted to compare two books to see which one was greater amount of `pages`, you could use the `__gt__` and the `__lt__` methods.

6. Now Let's add a `Library` class to our `library_tools/library.py` file. This is where we'll write our `Library` class. Let's create it below.

```python
class Library:
    def __init__(self, name):
        self.name = name
        self.books = []

    def __str__(self):
        return f"{self.name}"
```

- This is just the blue print for a library. It has a name and a list of books.
- Let's import our `Library` class into our `library_app.py` file and create a library. We can do this by adding the following line to the top of our file.
    - Let's also create a library and remove all of the code that `print`s attrbutes about the book. Your `library_app.py` file should look like this:

```python
from library_tools.library import Library
from library_tools.book import Book

if __name__ == '__main__':
    print("Welcome to our library App")
    print("--------------------------")
    # create a library
    library = Library("Edmonton Public Library")
    print(library)

    book = Book("The Lord of the Rings", "J.R.R. Tolkien", 1000)
```

- so you can see here that we imported the `Library` class the same way that we imported the `Book` class and also initialized it the same way (except with different arguments).
- now if you run the program you'll get output that looks like this.

```
$ python library_app.py
Welcome to our library App
--------------------------
Edmonton Public Library
```

- Now we have a library, but we don't have any books in it. Let's add a method to our `Library` class that will allow us to add books to our library.

6. Create two functions to add and remove books from our library. Let's add them to our `Library` class in the `library_tools/library.py` file.

```python
class Library:
    def __init__(self, name):
        self.name = name
        self.books = []
    def __str__(self):
        return f"{self.name}"

    # allows users to add books to our library
    def add_book(self, book):
        self.books.append(book)

    # allows users to list all the books in our library
    def list_books(self):
        print("Current books in our library:")
        if len(self.books) == 0:
            print("No books in our library")
        for book in self.books:
            print(F"- {book}")
```

- Note that here in the `add_book` method we're using `self.book.append(book)` to add a book to our list of books.
    - This works because the `self.books` was created in the `__init__` method, which is the constructor of the class.
- In the `list_books` we're going to loop through all of the books in our library and print them out.
    - You'll see that we're using the `__str__` method that we created in the `Book` class to print out the book.
- Now let's add some code to our `library_app.py` file to use these methods.

```
from library_tools.library import Library
from library_tools.book import Book

if __name__ == '__main__':
    print("Welcome to our library App")
    print("--------------------------")
    # create a library
    library = Library("Edmonton Public Library")
    print(library)
    # let's call the list books to observe that we have no books
    library.list_books()

    book = Book("The Lord of the Rings", "J.R.R. Tolkien", 1000)
    library.add_book(book)

    # let's call the list books to see that we have one book!
    library.list_books()
```

- Let's go observe the output to see what happens here.

```
$ python library_app.py
Welcome to our library App
```

```
--------------------------
Edmonton Public Library
Current books in our library:
No books in our library
Current books in our library:
- The Lord of the Rings by J.R.R. Tolkien
```

- So you can see that we have no books in our library, then we add a book to our library, and then we have one book in our library.

7. Let's add a few more books to our library so that we can see how this works with more than one book. Let's add the following code to our `library_app.py` file.

```python
from library_tools.library import Library
from library_tools.book import Book

if __name__ == '__main__':
    print("Welcome to our library App")
    print("--------------------------")
    # create a library
    library = Library("Edmonton Public Library")
    print(library)
    # let's call the list books to observe that we have no books
    library.list_books()

    # a few books
    book = Book("The Lord of the Rings", "J.R.R. Tolkien", 1000)
    bookTwo = Book("The Wheel of Time", "Robert Jordan", 690)
    bookThree = Book("The Way of Kings", "Brandon Sanderson", 1200)
    bookFour = Book("Mistborn", "Brandon Sanderson", 640)

    # add the books to our library
    library.add_book(book)
    library.add_book(bookTwo)
    library.add_book(bookThree)
    library.add_book(bookFour)

    # let's call the list books to see that we have one book!
    library.list_books()
```

- Now if you take a look at the output here we can see that we have all of the books in our library.

```
$ python library_app.py
Welcome to our library App
--------------------------
Edmonton Public Library
Current books in our library:
No books in our library
Current books in our library:
- The Lord of the Rings by J.R.R. Tolkien
- The Wheel of Time by Robert Jordan
- The Way of Kings by Brandon Sanderson
- Mistborn by Brandon Sanderson
```

Conclusion

This shows some of the fundamentals of class and objects in classes. Here's what we've learned a bit about:

- How to create a class
- How to create an instance of a class
- How to add attributes to a class
- How to add methods to a class
- How to use dunder methods to change the behaviour of a class.