By: kevin and hafza

Entertainment: strategy behaviour design pattern

This program allows users to interact with and specify various forms of entertainment and using the strategy behaviour design pattern.

The user chooses the service they want by specifying a type, genre, and mode of interaction.

- Type: the medium on which the user would make new purchases to add entries, being hard-copy (physical) or digital
- Genre: the genre of the service being used, between nonfiction or fiction
- Interact: the way in which the user will interact with the service. This can be chosen between “watch” for a movie, “read” for a book, or “listen” for music.

For “.service” the options to set the type,genre,interaction are:
HardCopy/Digital, Fiction/Non-Fiction, Read/Watch/Listen.

```
#enter the type, genre, interact for the service
Hobby.Service(HardCopy(),Fiction(),Read())

#displays the genre type
Hobby.GenreType()
#displays the Interaction type
Hobby.Int_Type()
#displays the entertainment type your buying
Hobby.CopyType()
```

The type can be changed through the setters.

```
#changing the genre type
Hobby.genre= Non_Fiction()
#displays the genre type
```

```
Hobby.GenreType()
```

Output would look like:

```
Entertainment is ready now
Genre is Fiction entertainment
Reading a book!
Buying a hard-copy

 Changing genre using setter:
Genre is Non-Fiction entertainment
```

The Entertainment class has getters & setters for each property (type/genre/interact) of the service.

```
class Entertainment:
    '''
    Base class: Entertainment types
    '''

    def __init__(self) -> None:
        self._buying = None
        self._genre = None
        self._interacting = None

    @property
    def buying(self):
        return self._buying

    @buying.setter
    def buying(self, buying):
        self._buying = buying
```

```
    @property
    def genre(self):
        return self._genre

    @genre.setter
    def genre(self, genre):
        self._genre = genre

    @property
    def interacting(self):
        return self._interacting

    @interacting.setter
    def interacting(self, interacting):
        self._interacting = interacting
```

The service itself is then defined using the properties as defined in their setters.

```
def Service(self, buying,genre,interacting=None):
    self._buying = buying
    self._genre = genre
    self._interacting = interacting
    print("Entertainment is ready now")

def GenreType(self):
    self._genre.genreType()

def CopyType(self):
    self._buying.buy()

def Int_Type(self):
    self._interacting.interact()
```