Writing Classes That Interact

You already know how to create your own types. Now, you will learn how to create two types that are related to each other.

Class Song

We have already created a class `Song`. As shown in the constructor, a `Song` has an artist, a title, and a duration in minutes and seconds. Our `Song` class also has an `__str__` method that prints the name of the artist, the title, and duration of the song.

```
class Song:
    """A song."""

    def __init__(self, artist, title, minutes, seconds):
        """ (Song, str, str, int, int) -> NoneType

        A Song with an artist, title, minutes, and seconds.

        >>> song = Song('Neil Young', 'Harvest Moon', 5, 3)
        >>> song.artist
        'Neil'
        >>> song.title
        'Harvest Moon'
        >>> song.minutes
        5
        >>> song.seconds
        3
        """

        self.title = title
        self.artist = artist
        self.minutes = minutes
        self.seconds = seconds

    def __str__(self):
        """ (Song) -> str

        Return a string representation of this song.

        >>> song = Song('Neil Young', 'Harvest Moon', 5, 3)
        >>> str(song)
        'Neil Young, Harvest Moon (5:03)'
        """

        return self.artist + ', ' + self.title + ' (' + str(self.minutes) \
            + ':' + str(self.seconds).rjust(2, '0') + ')'

if __name__ == '__main__':

    s1 = Song("Neil Young", "Harvest Moon", 5, 3)
    s2 = Song("Serena Ryder", "Stompa", 3, 15)

    print(s1)
    print(s2)
```

If we run the above module, here is the output:

```
>>>
Neil Young, Harvest Moon (5:03)
```

```
Serena Ryder, Stompa (3:15)
>>>
```

Class `Playlist`

Now, we want to create a new class called `Playlist` that will keep track of a playlist of songs. Initially, the `Playlist` object will have a title, but will not contain any songs. Therefore, the constructor will only take as input the name of the `Playlist`. The constructor will create an empty list for the songs. Next, we create an `add` method that adds songs to the `Playlist`. Here is the code so far:

```
import song

class Playlist:

    def __init__(self, title):
        """ (Playlist, str) -> NoneType

        >>> playlist = Playlist('Canadian Artists')
        >>> playlist.title
        'Canadian Artists'
        >>> playlist.songs
        []
        """

        self.title = title
        self.songs = []

    def add(self, song):
        """ (Playlist, Song) -> NoneType

        Add song to this playlist.

        >>> stompa = song.Song("Serena Ryder", "Stompa", 3, 15)
        >>> playlist = Playlist('Canadian Artists')
        >>> playlist.add_song(stompa)
        >>> playlist.songs
        [stompa]
        """

        self.songs.append(song)

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

Notice that we `import song`, so that we can use class `Song` it in our new class. If we run the above module, we will get no output, which means that the `doctest` passed. Now, let us create a new method that returns the total duration of all the `Song` objects in the `Playlist`. The return type of the `get_duration` method will be a tuple of minutes and seconds. Here is the code for `get_duration`:

```
def get_duration(self):
    """ (Playlist) -> (int, int)

    Return the duration of this playlist as tuple of minutes and
    seconds.

    >>> playlist = Playlist('Canadian Artists')
    >>> playlist.add(song.Song('Neil Young', 'Harvest Moon', 5, 3))
    >>> playlist.add(song.Song('Serena Ryder', 'Stompa', 3, 15)
    >>> playlist.duration()
    (8, 18)
    """

    total_minutes = 0
```

```
    total_seconds = 0

    for song in self.songs:
        total_minutes = total_minutes + song.minutes
        total_seconds = total_minutes + song.seconds

    return (total_minutes,  total_seconds)
```

We use two accumulators: one to keep track of the sum of the minutes for all songs in the playlist, and another one to keep track of the sum of the seconds for all songs in the playlist. We look up a song's length using its `seconds` and `minutes` instance variables. The `for` loop loops over all the `song` objects in the playlist (in the `songs` instance variable), and for each `Song` object, its `minutes` and `seconds` are added to the appropriate accumulator.

Once the loop has finished executing, the total number of minutes and seconds is returned. However, it is possible for the total number of seconds for all the songs to be greater than 60. For instance, assume a song that is 3:35 (3 minutes and 35 seconds) long, and another song that is 2:50 (2 minutes and 50 seconds) long. If we simply add these two songs just as the above `get_duration` method does, we will get a tuple (5, 85) or 5 minutes and 85 seconds. However, this is not a nice way to report the total duration of the songs. The correct thing to do is to return (6, 25). This can be acheived by removing 60 seconds from the seconds accumulator, and adding it to the minutes accumulator. We'll update the expression returned by the method:

```
def get_duration(self):
    """ (Playlist) -> (int, int)

    Return the duration of this playlist as tuple of minutes and
    seconds.

    >>> playlist = Playlist('Canadian Artists')
    >>> playlist.add(song.Song('Neil Young', 'Harvest Moon', 5, 3))
    >>> playlist.add(song.Song('Serena Ryder', 'Stompa', 3, 15)
    >>> playlist.duration()
    (8, 18)
    """

    total_minutes = 0
    total_seconds = 0

    for song in self.songs:
        total_minutes += song.minutes
        total_seconds += song.seconds

    return (total_minutes + total_seconds // 60,  total_seconds % 60)
```

Finally, we will write a `__str__` method in order to have a nice output when the `print` function is called on our `Playlist` object:

```
def __str__(self):
    """ (Song) -> str

    Return a string representation of this playlist.

    >>> playlist = Playlist('Canadian Artists')
    >>> playlist.add(song.Song('Neil Young', 'Harvest Moon', 5, 3))
    >>> playlist.add(song.Song('Serena Ryder', 'Stompa', 3, 15)
    '''Canadian Artists (8:18)
    Neil Young, Harvest Moon (5:03)
    Serena Ryder, Stompa (3:15)'''
    """

    duration = self.get_duration()
    minutes = str(duration[0])
    seconds = str(duration[1]).rjust(2, '0')
```

```
    result = self.title + ' (' + minutes + ':' + seconds + ')'

    # Append the songs in the playlist.
    song_num = 1
    for song in self.songs:
        result = result + '\n' + str(song_num) + '. ' + str(song)
        song_num = song_num + 1

    return result
```

In the above `__str__` method, we first gather general data about the playlist, and then move on to get information for each individual song by calling `str`, which uses `Song`'s `__str__` method. Notice that for the string representation of `seconds`, we perform `rjust`. Here is an example of how we'll use it:

```
>>> '3'.rjust(2,'0')
'03'
>>> '14'.rjust(2, '0')
'14'
```

Essentially, `rjust` makes sure that the string has exactly two characters. If it doesn't, then `rjust` will add as many `0`s as necessary to the beginning of the string in order to make its length 2. Finally, in `__str__`, we create a variable `result` that will contain the name of the playlist and the total duration of all the songs in the playlist. Next, we add the name and duration of each individual song to the `result` variable. Notice that we keep of track of the number of songs with the variable `song_num`; this varialbe will be used for the string representation of the `Playlist` object. Finally, `__str__` will return the `result` string we created.

The last thing we will do is create our main program:

```
if __name__ == '__main__':

    playlist = Playlist('Canadian Artists')
    playlist.add(song.Song("Neil Young", "Harvest Moon", 5, 3))
    playlist.add(song.Song("Serena Ryder", "Stompa", 3, 15))
    playlist.add(song.Song("Stompin' Tom Connors", "The Hockey Song", 2, 17))

    print(playlist)
```

If we run the `Playlist` module, the output will be:

```
>>>
Canadian Artists (10:35)
1. Serena Ryder, Stompa (3:15)
2. Neil Young, Harvest Moon (5:03)
3. Stompin' Tom Connors, The Hockey Song (2:17)
>>>
```

Jennifer Campbell • Paul Gries
University of Toronto
