Classes and Objects

Why is this important?

Classes and objects can give you group function

Steps

1. Take a look at `data/course_data.py` and observe the data in that file. You can see here that we're going to be working with a list of dictionaries to create some objects.

2. Let's create a file called `tools/course.py`. This file will contain a class called `Course`. Add the following code to the file:

```
class Course:
    def __init__(self, name):
        self.name = name
        self.students = []
        self.assignments = []

    def __str__(self):
        return f"{self.name} has {len(self.students)} students and
{len(self.assignments)} assignments"
```

- Note: the attribute `student` will be of type `List[Student]` and the attribute `assignments` will be of type `List[Assignment]`. We'll be creating these in the next steps.

3. Let's create a file called `tools/student.py`. This file will contain a class called `Student`. Add the following code to the file:

```
class Student:
    def __init__(self, id, name):
        self.id = id
        self.name = name
        self.submissions = []

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

- Note: the attribute `submissions` will be of type `List[Submission]`.

1. Now let's create a file called `tools/assignment.py`. This file will contain a class called `Assignment`. Add the following code to the file:

```
class Assignment():
    def __init__(self, id, name):
        self.id = id
        self.name = name

    def __str__(self):
        return f"Assignment ID: {self.id}, Assignment Name: {self.name}"

    def __repr__(self):
        return str(self)
```

5. Now Let's create a submission class. This file will contain a class called `Submission`. Add the following code to the file:

```
class Submission:
    def __init__(self, student, assignment, grade):
        self.student = student
        self.assignment = assignment
        self.grade = grade

    def __str__(self):
        return f"{self.student.name} received {self.grade} on
{self.assignment}"
```

5. Let's use our classes to create some objects. In the `course_app.py` file add the following code:

```
from data.course_data import student_data, assignment_data,
submission_data

from tools.course import Course

if __name__ == "__main__":
    # Create course
    course = Course("Dans Basketball Mastery Course")
    # display the course information.
    print(course)
```

- now if you run the `course_app.py` file you should see the following output:

```
$ python course_app.py
Dans Basketball Mastery Course has 0 students and 0 assignments
```

5. Let's create the functionality to add students to the `Course` class and add those objects in the `course_app.py` file. Add the following code to the file.

- First, let's add the following method to the course class in `tools/course.py` file:

```
class Course:
    def __init__(self, name):
        self.name = name
        self.students = []
        self.assignments = []

    def __str__(self):
        return f"{self.name} has {len(self.students)} students and
{len(self.assignments)} assignments"

    # new method tp add students.
    def add_student(self, student):
        self.students.append(student)
```

- Second, let's create an `add_student` method in the `course_app.py` file which will add student objects by the `Student` class to the course object. Add the following code to the `course_app.py` file:

```
from data.course_data import student_data, assignment_data,
submission_data

from tools.course import Course
from tools.student import Student

def add_students(course):
    for student in student_data:
        student_instance = Student(student["id"], student["name"])
        course.add_student(studentInstance)


if __name__ == "__main__":
    # Create course
    course = Course("Dans Basketball Mastery Course")

    # Add students to course
    add_students(course)

    # display course information
    print(course)
```

- Now you'll see that the course has 3 students, take a look at the output below:

```
$ python course_app.py
Dans Basketball Mastery Course has 3 students and 0 assignments
```

6. Let's create the functionality to add assignments to the `Course` class and add those objects in the `course_app.py` file. Add the following code to the file.

- First, let's add the following method to the course class in `tools/course.py` file.

```
class Course:
    def __init__(self, name):
        self.name = name
        self.students = []
        self.assignments = []

    def __str__(self):
        return f"{self.name} has {len(self.students)} students and
{len(self.assignments)} assignments"

    def add_student(self, student):
        self.students.append(student)

    # new method to add assignments.
    def add_assignment(self, assignment):
        self.assignments.append(assignment)
```

- Second, let's create an `add_assignments` method in the `course_app.py` file which will add assignment objects by the `Assignment` class to the course object. Add the following code to the `course_app.py` file:

```
# other imports.
from tools.assignment import Assignment

# ... previous add_students code ...

def add_assignments(course):
    for assignment in assignment_data:
        assignment_instance = Assignment(assignment["id"],
assignment["name"])
        course.add_assignment(assignment_instance)

# if __name__ == "__main__": ... previous code ...
```

- Now you'll see that the course has 3 students and 2 assignments, take a look at the output below:

```
$ python course_app.py
Dans Basketball Mastery Course has 3 students and 2 assignments
```

7. Let's create the functionality to add submissions to the `Course` class and add those objects in the `course_app.py` file. Add the following code to the file.

- First, let's create a couple of helper methods in `Course` class so that we can get students from the course and get assignments from the course.

```python
class Course:
    # ... other methods ...
    def get_student(self, student_id):
        for student in self.students:
            if student.id == student_id:
                return student
        return None

    def get_assignment(self, assignment_id):
        for assignment in self.assignments:
            if assignment.id == assignment_id:
                return assignment
        return None
```

- Second, let's create a method in the `Student` class so that a student can have submissions.

```python
class Student:
    # ... other methods ...
    def add_submission(self, submission):
        self.submissions.append(submission)
```

- Third, let's create a method on `add_submissions` method in the `course_app.py`. We'll be creating the `Submission` objects in this method and adding them to a "student instance".

```python
# ... other imports ...
from tools.submission import Submission

# ... other methods ...
def add_submissions(course):
    for submission in submission_data:
        student = course.get_student(submission["student_id"])
        assignment = course.get_assignment(submission["assignment_id"])
        submission = Submission(student, assignment, submission["grade"])
        student.add_submission(submission)

if __name__ == "__main__":
    # Create course
    course = Course("Dans Basketball Mastery Course")

    # Add students to course
    add_students(course)

    # Add assignments to course
    add_assignments(course)
    # Add submissions to the course
    add_submissions(course)

    # display course information
    print(course)
```

- Once we have this add a `breakpoint()` to the `course_app.py` file right under the `print(course)` line. Run the file and you can see actually see the relationship between the course, students, and submissions by writing `[submission.grade for submission in course.students[0].submissions]` in the debugger. Take a look at the output below.

```
$ python course_app.py
Dans Basketball Mastery Course has 3 students and 2 assignments
--Return--
> c:\users\dmouris\sdev1001-master-course\classes-and-
objects\course_example_start\course_app.py(44)<module>()->None
-> breakpoint()
(Pdb) [submission.grade for submission in course.students[0].submissions]
[100, 90]
(Pdb)
```

- You can see that you can traverse the course object to get the students and then the submissions for each student using the attributes here.

9. Let's create a course average method on the `Course` class that will leverage traversing the students and submissions to get the average grade for the submission. - Add the following code to the `tools/course.py` file.

```
class Course:
    # ... other methods ...

    def get_course_average(self):
        total = 0
        number_of_submissions = 0

        for student in self.students:
            for submission in student.submissions:
                total += submission.grade
                number_of_submissions += 1

        return total / number_of_submissions
```

- Let's add the following code to the `course_app.py` file to display the course average.

```
# ... other imports ...
```

```python
if __name__ == "__main__":
    # Create course
    course = Course("Dans Basketball Mastery Course")

    # ... adding students, assignments, and submissions ...
    # display course information
    print(course)

    # display course average
    print(F"Course average: {course.get_course_average()}")
```