Classes

Announcements

Class Statements

Class Statements Create User-Defined Types

A class statement creates a new type

Idea: All students have a name and a midterm score mt1_points; the Student class should add those attributes to each newly created instance

The **@dataclass** decorator simplifies class statements

```
from dataclasses import dataclass

@dataclass
class Student:
    name: str
    mt1_points: int


>>> eve = Student("Eve", 26)
```

A class can describe the behavior of its instances

Idea: All students share a total_points method and a assign_mt1_grade method

```
>>> bao = Student('Bao', in_mega=False)
>>> bao.total_points()
0
>>> bao.assign_mt1_grade(26)
>>> bao.total_points()
20
```

The Student Class

```
class Student:
    def __init__(self, name, in_mega):
        self.name = name
        self.lab_points = 0
        if in_mega:
            self.lab_points = 10
        self.mt1_points = 0
        self.a_plus_points = 0

    def total_points(self):
        return self.lab_points + self.mt2_points

    def assign_mt1_grade(self, points):
        self.mt1_points = min(20, points)
        if points == 26:
            self.a_plus_points += 5
```

__init__ is a special method name for the function that constructs a Student instance

self is the instance of the Student class on which total_points was invoked

Methods are functions defined in a class statement

bao is an ***instance*** of the Student class

```
>>> bao = Student('Bao', False)
>>> eve = Student('Eve', False)
```

mt1_points is an ***attribute***

```
>>> bao.mt1_points
0
```

total_points() is a ***method***

```
>>> bao.total_points()
0
>>> bao.assign_mt1_grade(26)
>>> bao.mt1_points
20
>>> eve.mt1_points
0
```

Discussion Question: Midterm Retakes

```
class Student:
    def __init__(self, name, in_mega):
        self.name = name
        self.lab_points = 0
        if in_mega:
            self.lab_points = 10
        self.mt1_points = 0
        self.a_plus_points = 0

    def retake_mt1(self, points):
        """ Replace MT1 grade with points, if points is higher than the existing MT1 grade.

        Cap mt1_points at 20, even if points is higher.
        """
        if ____________________________:

            ____________________________________
```

Discussion Question: Midterm Retakes

```
class Student:
    def __init__(self, name, in_mega):
        self.name = name
        self.lab_points = 0
        if in_mega:
            self.lab_points = 10
        self.mt1_points = 0
        self.a_plus_points = 0

    def retake_mt1(self, points):
        """ Replace MT1 grade with points, if points is higher than the existing MT1 grade.

        Cap mt1_points at 20, even if points is higher.
        """
        if points > self.mt1_points:
            self.mt1_points = min(points, 20)
```

Discussion Question: Create Many Students

Write a function **create_students** that takes a list of strings called **regular** that has all of the students in regular lab, and **mega** that has all of the students in mega lab. It returns a dictionary in which each name is a key, and its value is a new **Student** with that name.

```
def create_students(regular: list[str], mega: list[str]) -> dict[str, Student]:
    """
    >>> students = create_students(["Daniel", "Darren", "Peyton"], ["Terry", "Tejas"])
    >>> students["Darren"].lab_points
    0
    >>> students[“Terry”].lab_points
    10
    """
    result = {_____________________________________________}
    for name in ________:
        _____________________________________
    return result
```

dictionary

A Student instance

Discussion Question: Create Many Accounts

Write a function **create_students** that takes a list of strings called **regular** that has all of the students in regular lab, and **mega** that has all of the students in mega lab. It returns a dictionary in which each name is a key, and its value is a new **Student** with that name.

```
def create_students(regular: list[str], mega: list[str]) -> dict[str, Student]:
    """
    >>> students = create_students(["Daniel", "Darren", "Peyton"], ["Terry", "Tejas"])
    >>> students["Darren"].lab_points
    0
    >>> students["Terry"].lab_points
    10
    """
    result = {name: Student(name, False) for name in regular}
    for name in mega:
        result[name] = Student(name, True)
    return result
```

dictionary

A Student instance

Animation Demos by Hany Farid