CSC108H Lecture 24

Dan Zingaro

November 7, 2012

Students in Lists

- Suppose we want to store students' names, their year, and their most recent three course marks
- Perhaps we could use a nested list

```
students = [
            ['Dan', 4, [80, 82, 6]],
            ['Joe', 2, [49, 48, 82]]
           ]
```

However:

- We are imposing unnecessary order on the elements (why is year second and marks third?)
- The elements do not have names
- No convenient way to print students
- Not easy to support multiple types of students (e.g. first-year students may have no prior course marks)

Students in Dictionaries

- If we use a nested dictionary instead, we can use strings instead of indices to refer to the students
- However, still no flexibility to support different kinds of students
- Also no way to ask a student to "do something" (like calculate average mark)

```
students = {
         'Dan':{'name':'Dan', 'year':4, 'marks':[80, 82, 6]},
         'Joe':{'name':'Joe', 'year':2, 'marks':[49, 48, 82]}
           }
```

Classes and Methods

- Sometimes, the built-in objects are not natural for supporting new types of data
- By defining a new **class**, we can add a new type to Python
- We can then make objects of that type (e.g. objects of type `Student`)
- Class names are nouns: student, course, rectangle, animal, ship, (and str, list, dict!)
- Methods are actions specific to each type of object
  - e.g. for ship: move, turn, shoot, dodge, raise shields, lower shields, teleport

Attributes

- An **attribute** is a feature or characteristic of an object
- Unlike a method, it is not an action
- An attribute is something that an object has, not something the object does

Class name: `Ship`
Possible methods: `move`, `turn`, `shoot`, `dodge`, `raise_shields`, `lower_shields`, `teleport`
Possible attributes: `weight`, `length`, `width`, `direction`, `num_bullets`

ConcepTest

Which of the following is **not** a possible method for a `Car` class?

- A. `open_window`
- B. `accelerate`
- C. `num_wheels`
- D. `turn_right`
- E. `apply_brakes`

ConcepTest

Which of the following is **not** a possible method for a `Person` class?

- A. `hair_colour`
- B. `climb_stairs`
- C. `speak`
- D. `walk`
- E. `jump`

Creating Objects

- When an object is created, its `__init__` method is called
- `__init__` is known as a **constructor** because it constructs objects
- Inside `__init__`, assign values to the object's attributes
- For example, consider a `Point` class whose objects store x- and y-coordinates for a two-dimensional point
- The two required attributes are `x` and `y`

Creating Objects...

```
class Point(object):
  '''Two-dimensional points'''

  def __init__ (self):
    '''() -> Point
    Create two-dimensional point
    '''
    self.x = 0
    self.y = 0
```

- Don't forget the `self.` before each attribute
- In all methods you write, `self` means "current object"
- You can create a point using e.g. `p = Point()`
- `p.x` and `p.y` access the attributes of `p`

ConcepTest

What is the output of this code?

```
p1 = Point()
p1.x += 2
p1.y += 3
p2 = Point()
p2.x += 4
print(p2.x, p2.y)
```

- A. 4 0
- B. 0 0
- C. 7 3
- D. 2 3

Creating Objects with Parameters

A `Point` that always starts at $(0, 0)$ may not be as useful as a `Point` that can start at a specified location.

```
class Point(object):
  '''Two-dimensional points'''

  def __init__ (self, x, y):
    '''(int, int) -> Point
    Create two-dimensional Point at (x, y)
    '''
    self.x = x
    self.y = y
```

- Now, you create a `Point` using e.g. `p = Point(2, 5)`
- `p.x` and `p.y` then have these initial values