CS2110 Lecture 23 Mar. 17, 2021

- DS 6 due Friday
- HW 6 available, due Thursday, 3/25

Last time

- Introduce classes and object oriented programming (Ch 17, 18, 19)
  - Demonstrate use of classes as simple containers of properties (classes with no methods) - an alternative representation for data to lists and dictionaries
  - time1.py

Today

- Using classes in a more "object-oriented" way, where methods provide the "API" for interacting with objects

Next time

- class attributes (not in interactive text but in 18.2 of pdf version of text)
- Ch 19: inheritance

Classes with methods (the more "object-oriented" way)

- General rule for defining classes:
  - **always** define an __init__ method initializing values for all properties/attributes (e.g. hour, minutes, seconds for Time)
  - define methods that represent the "public interface" to the class. Users should work with instances of the class only via these methods rather than by accessing object attributes directly. First argument to a method is always the object that invokes it. Standard practice is to use variable name 'self'

__init__ methods and “constructors”

```
class Time:
    def __init__ (self, hour = 0, minutes = 0, second = 0):
        self.hour = hour
        self.minutes = minutes
        self.seconds = seconds

>>> t1 = Time(3, 24, 59)
>>> t.hour
3
>>> t.seconds
59
```

HOW DOES THIS WORK??

When you create an object using a “constructor”: e.g. Time(…)

1. Python first creates empty object
2. Passes that empty object to __init__ with any additional arguments provided to constructor
3. returns the new object (even though there is no “return” line in init)

Make things look nice using __repr__ and/or __str__ methods

```
class Time
        def __init__(. . .):
                . . .
        def __repr__(self):
                return "Time({}, {}, {})".format(
                                self.hour, self.minutes, self.seconds)
        def __str__(self):
                ampm = "AM" if self.hour <12 else "PM"
                return "{:02d}:{:02d}:{:02d} {}".format(
                                self.hour%12,self.minutes,self.seconds, ampm)
```

```
>>> t = Time(10,23,59)
>>> t
Time(10,23, 59)
>>> print(t)
10:23:59 AM
>>>  str(t)
“10:23:59 AM”
```

__repr__ and __str__ methods: used to define how object displays or gets converted to string. Many Python programmers don’t know the distinction between the two. You don’t need to know. If you’re only going to define one, define __repr__. However, many people argue that best practice is: __repr__ should produce string that is what you would type in to create object similar object, while __str__ should simply yield a nice “readable” form.

Notes on development of classes

- Look at implementation of
  - incrementTime(self)
  - laterTime(self)

  methods in time2.py. Same basic code as in time1.py but now in OO style. First argument to a method is always object that invokes the method, and standard practice is to use var name 'self'

- Nice feature of classes: you can **overload** operators. That is, you can define how +, -, <, etc. apply to objects of classes that you define
  - __add__ for + (and __radd__)
  - __lt__ for <
  - __eq__ for ==, etc.

  See how these are used in time2.py

Notes on development of classes

- AGAIN, best practice as a user of class is avoid directly accessing object attributes. I.e. when you have a time object t, don't use t.hour. Use only methods. WHY?
- If we only use methods, the class developer can change in the internal representation (maybe to make things more efficient). E.g instead of using three attributes – hour, minutes, seconds - to represent time in the Time class, could just use seconds! Can still make all the methods work the same, print in human friendly form, etc. implementation. See time2Alt.py

Other basic class examples

catdog.py: Each class has

- a simple constructor (with optional name as argument, and default if nothing provided)
- a __repr__ so objects will display readably
- a few methods: speak, setName, getName,fetch (only Dog)

```
>>> c1 = Cat()
>>> c2 = Cat()
>>> d = Dog("spot")
>>> for animal in [c, c2, d]:
                print(animal.getName())
                animal.speak()
fluffy
meow
fluffy
meow
spot
woof
```

testCatDog()

circle.py: study this class carefully. Use similar style for DS6 Rect problem and HW6 Box problem

HW 6 hints

- Q1: think carefully about overlaps – draw pictures
  - Think dimension by dimension – three 1D problems
    - if they don't overlap in x, they don't overlap
      - » Express this in terms of center x's and half-widths
    - if ..
    - if ..
  - _____ ----------
- Q2
  - Ensure legal moves – i.e. if user enters an illegal choice, print something appropriate and ask for a new choice.
  - Computer gameplay can be random (but must be legal). You can use, for instance, random.randint(…)and random.choice(…) to choose (non-zero) number of balls and (non-empty) heap. (also fine, of course, if you make yours smarter than random)
- Q3 is very very easy compared to Q1 and Q2

Next time

Finish our quick look at object-oriented programming:

- class attributes – not in interactive text (but in 18.3 of pdf of non-interactive text)
- Ch 19 – inheritance