CS1010S Tutorial 9

Wang Zexin

National University of Singapore
Year 2 QF

wang.zexin@u.nus.edu

April 2, 2017

Today's Agenda

1. Recap
2. Question One
   - Standard solutions
3. Question 2
   - Standard solutions
4. Question 3
5. Question 4
   - Part 1
6. Extra stuff: Two simple final questions
   - A simple final question on Message Parsing
   - A simple final question on OOP

Recap - OOP

- What are classes and objects?
- Blueprints/models
- Initialization
  - *__init__*
  - *self.what = what*
- Encapsulation
  - Your attributes should invisible.
  - Use secret attribute names.
  - You should be assigning getters for those which should be public.
  - *self.__name* is somehow 'secret'
  - *getName()* method is public!
- Inheritance
  - No need to redefine attributes and methods for subclasses.
  - Format: *classA(B)* :
  - Call superclass's method: *super.()__init__()*

Recap - OOP

- Multiple Inheritance
  - Format: $classA(B, C)$:
  - All the attributes and methods inherited
  - What if there are two different methods in both superclasses?
  - Use B
  - Which one is called when you call for $super()$?
  - B
- Polymorphism through Inheritance!
  - Different methods with the same name
  - Overriding methods: rewrite methods
  - Overriding attributes: rewrite constructor
  - You can add more content after calling the superclass method!
  - Overriding operators: rewrite $\_\_eq\_\_$ / $\_\_add\_\_$
- To check the attributes/methods $dir(objectName)$
- To check the class of object: $isinstance(objectName, className)$

Recap - Dictionary

A new data structure

- keys and values - iterable objects in **order**
- construct a dictionary
  - dict([(1,2),(2,4),(3,6)])
  - dict()
  - {'a':1, 'b':2}
- check for a key in dictionary:
  - d.get(key) - None if no such key
  - if d[key]: - may have key error
  - if key in d:
  - if key in d.keys():
- del d[key]
- d.clear()

Question 1

Implement the class *Thing* such that it satisfies the properties and methods below.

- The constructor should take in 1 parameter, the name of the *Thing*.
- owner : an attribute that stores the owner object of the *Thing*.
- *is_owned*(): returns a boolean value, *True* if the thing is "owned" and *False* otherwise.
- *get_owner*(): returns the *Person* object who owns the *Thing* object.

Question 1 Discussion

How should we define this *Thing* class?

- The constructor only needs to take care of two attribute.
- Take in name and store as name.
- Set owner to None first.
- Define *is_owned*() in the same way as a normal predicate.
- *get_owner*(): just return the owner attribute.

Question 1 Zexin's solution

```
class Thing:
    def __init__(self, name):
        self.name = name
        self.owner = None

    def is_owned(self):
        return self.owner != None

    def get_owner(self):
        return self.owner
```

Question 2

Modify and extend your *Thing* definition in Question 1.

- *get_name*() : returns the *name* (string) of the *Thing*.
- *place* : Just like the *owner* attribute, we need to keep state of the *Place*.
- *get_place*() : returns the place associated with the *Thing*.

Question 2 Discussion

Your *Thing* should include:

- Definition for place in the constructor
- Getter for name
- Getter for place

Question 2 Zexin's solution

```
class Thing:
    def __init__(self, name):
        self.name = name
        self.owner = None
        self.place = None

    def is_owned(self):
        return self.owner != None

    def get_owner(self):
        return self.owner

    def get_name(self):
        return self.name

    def get_place(self):
        return self.place
```

You can use isinstance to check if necessary.

Question 3

What is wrong with this code?

```
class Thing(MobileObject):
    def __init__(self, name):
        self.name = name
        self.owner = None
        # Superclass constructor is not called!
        # Hence place attribute is not inherited!

    def is_owned(self):
        return self.owner != None

    def get_owner(self):
        return self.owner

    def get_name(self):
        return self.name

    def get_place(self):
        return self.place
```

Question 3 Discussion

Note that what happened are:

- The constructor has been overriden.
- The attribute *place* is never initiated.
- Still the user is trying to call *get_place*
- Exception will happen . . .

Question 4 Part 1

Draw simple inheritance diagram showing all the kinds of objects (classes) defined in the adventure game system, the inheritance relations between them, and the methods defined for each class. This is critical in helping you to understand the OOP system in *hungry_games.py* for your missions.

Question 4 Part 1 Discussion

Let Zexin open *engine.py* and *hungry_games.py* and dissect the classes.

- NamedObject
- Place
- MobileObject
- Thing
- SDCard
- LivingThing
- Person
- Troll

Question 4 Part 1 Solution

```
NamedObject [ name | get_name ]
|
+-Place [ objects, neighbor_dict | add_object, del_object, get_objects,
|         get_exits, add_neighbor, get_neighbors, get_neighbor_at ]
|
+-MobileObject [ place | get_place ]
  |
  +- Thing [ owner, ownable | get_owner, is_owned ]
  |  |
  |  SDCard [ id_num | is_sdcard, copy ]
  |
  +- LivingThing [ health, threshold | get_threshold, get_health, add_health,
     |             reduce_health, go_to_heaven, move_to, act ]
     |
     +- Person [ inventory | take, lose, go, say, look_around,
        |        get_inventory, objects_around, get_exits, str_to_thing]
        |
        +- Troll [ | act, eat_person ]
```

Question 4 Part 2

ice_cream = Thing("ice_cream")
ice_cream.owner = beng
Come up with statements whose evaluation will reveal all the properties of ice_cream and verify that its (new) owner is indeed beng.

Question 4 Part 2 Discussion

How to tackle this?

- Make sure that method/attribute exist before calling it!
- How to check for all the attributes and methods of an object?
- How to check for equivalence of objects?

Question 4 Part 2 Zexin's solution

```
ice_cream = Thing("ice_cream")
ice_cream.owner = beng
print(dir(ice_cream))
print(ice_cream.get_owner() is beng)
```

Question 4 Part 3

ice_cream = Thing("ice_cream")
ice_cream.owner = beng
beng.ice_cream = ice_cream
Is there anything wrong with the last two statements? What is the moral of the story?

Question 4 Part 3 Discussion

How to tackle this?

- It is highly possible that ice_cream.owner exists.
- But whether beng.ice_cream exists or not, we have no idea.
- Also, what is the purpose of doing this when encapsulation is not preserved?

Question 4 Part 4

ice_cream = Thing(“ice_cream”)
rum_and_raisin = NamedObject(”ice_cream”)
Are ice_cream and rum_and_raisin the same object (i.e., does ice_cream is rum_and_raisin evaluate to True)?

Question 4 Part 4 Discussion

How to tackle this?

- Refer to our discussion in the first tutorial . . .
- They have different storage address.

Question 4 Part 5

Write something so that we can evaluate == differently to judge whether two objects of Thing can be compared based on values instead of storage address.

Question 4 Part 5 Discussion

How to tackle this?

- Remember the overriding of methods?
- Let us override some operators!

Question 4 Part 5 Zexin's solution

```
class Thing(MobileObject):
    def __init__(self, name):
        super().__init__(name, None)
        self.owner = None
        self.ownable = True

    def get_owner(self):
        return self.owner

    def is_owned(self):
        return self.owner is not None

    def __eq__(self, other):
        return isinstance(other, Thing) and \
               self.get_name() == other.get_name() and \
               self.get_place() == other.get_place()
```

Extra stuff: A simple final question on Message Parsing

If time permits, we will go through this.

```
def top ():
    x = [0]
    def next(*args):
        op = args[0]
        if op == 'add':
            x[0] = x[0] + args[1]
            return x[0]
    return next

a = top()
a('add', 5)
a('add', 2)
```

- Remember the recipe for message parsing?
- The answer should be 7.

Extra stuff: A simple final question on OOP

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

    def get_name(self):
        return self.name

    def set_name(self, new_name):
        self.name = new_name

class PlayRobot(Robot):
    def __init__(self, name, color):
        super().__init__(name)
        self.color = color
        self.fun_factor = 0

    def play(self, time):
        self.fun_factor += time
        print("New fun factor: " + str(self.fun_factor))

class StudyRobot(Robot):
    def __init__(self, name):
        super().__init__(name)
        self.skills = 0

    def teach(self, time):
        self.skills += time
        print("New skills level:" + str(self.skills))
```

Extra stuff: A simple final question on OOP

After taking a look at Ermie's code, Ron complains that he would not be able to change the name of a `PlayRobot` instance, as there are no methods to handle that in the `PlayRobot` class definition. Do you agree with Ron? **Briefly explain** why or why not? [4 marks]

Recall inheritance!

This is giving mark.

Extra stuff: A simple final question on OOP

Ermie would like to define a new class `BuddyRobot`, that is both a `PlayRobot` and a `StudyRobot` (in that order). A `BuddyRobot` is special because it can sing! The input parameters to a `BuddyRobot` include its `name`, `color`, and a `song` (assumed to be a single string for now). A `BuddyRobot` has a method `sing` that prints out the lyrics (a string) of the `song`. Help Ermie complete the following definition for the class `BuddyRobot`. State any assumptions you make. [10 marks]

Recall multiple inheritance!

Extra stuff: A simple final question on OOP

Help Ermie define a new `teach` method for the `BuddyRobot` class by extending the `teach` method of `StudyRobot` with a new behavior – it will `sing` after teaching! State any assumptions you make. [6 marks]

Recall overriding and calling superclass methods.

Feedback & more

- Slides + relevant material available at:

  `https://github.com/wangzexin/Teaching`

- After the tutorial, if you have further questions:

  `wang.zexin@u.nus.edu`

Thank You

wang.zexin@u.nus.edu