CISS450: Artificial Intelligence
Lecture 16: Classes and Objects

Yihsiang Liow

Agenda

- Study Object-Oriented Programming concepts: classes, objects, attributes, methods
- Here are some buzzwords:
  - OO = object-oriented
  - OOA = object-oriented analysis
  - OOD = object-oriented design
  - OOP = object-oriented programming
- OO skills are extremely important in developing large scale software

Objects

- In real life, we tend to associate names to a bunch of basic things
- Example: Look at “John Doe was born 04/01/1970”. Think of John Doe as an entity. He has the following associated with him:
  - First name = “John”
  - Last name = “Doe”
  - Birthdate = “04/01/1970”
  - GPA = 3.5

Objects

- For this example, John Doe is an object with values “John”, “Doe”, “04/01/1970”, 3.5 for attributes first name, last name, birthdate and GPA respectively
- For C/C++ you can think of structure (or class):

```
struct Person
{
    char * firstname;
    char * lastname;
    char * birthdate;
    int gpa;
};
```

Objects

- For Python, you can think of dictionaries:

```
john_doe = {}
john_doe['firstname'] = 'John'
john_doe['lastname'] = 'Doe'
john_doe['birthdate'] = '04/01/1970'
john_doe['gpa'] = 3500
```

Attaching Functions to Objects

- Besides attaching “smaller” data to a larger concept/name, we also want to attach functions to these things.
- Example:

```
xs = [3,2,1]
xs.sort()
```

- In this example, `xs` has a function called `sort`

Class

- We abstract the common parts of objects and create a “stamp”. The stamp will describe all the attributes of objects that this stamp can create. Furthermore, this stamp will describe all the functions its objects can perform.
- This stamp is called a ***class***.
- The things created by this stamp are ***objects***. The “parts” of an object are called ***attributes***.
- The functions these objects can perform are called ***methods***

Class

- WARNING: In C/C++, object attributes are called member variables and methods called member functions.

Attribute Protection

- You want to protect the attributes of your objects so that they are not directly accessible.
- Why?
- First: Security
- Second: The internal representation might change in the future. (See example later).

First Example

- Now to create your own class. Here's a standard example ...
- Suppose we want to think about 2-dimensional points (or rather vectors). Think about the attributes and methods.
- Attributes: Each point should have two value, the x- and y-coord of the point

First Example

```
class vec2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def get_x(self):
        return self.x
    def get_y(self):
        return self.y

p = vec2D(1,3)
print(p.get_x(), p.get_y())
```

constructor

attributes: x, y

Calling constructor

p.get_x() is the same as vec2D.get_x(p). So self becomes p.

First Example

If you want outsiders to modify x and y attribute:

```
class vec2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def get_x(self): return self.x
    def get_y(self): return self.y
    def set_x(self, x): self.x = x
    def set_y(self, y): self.y = y
p = vec2D(1,3)
p.set_x(2)
print(p.get_x(), p.get_y())
```

Protecting Attributes

- Why do you want to protect the object attributes from outsider's direct access?
- Because you might want to change the way you represent vec2D objects internally:

```
class vec2D:
    def __init__(self, x, y):
        self.list = [x,y]
    def get_x(self): return self.list[0]
    def get_y(self): return self.list[1]
    def set_x(self, x): self.list[0] = x
    def set_y(self, y): self.list[1] = y
```

Protecting Attributes

- But Python does not prevent you from accessing the attributes
- Try:

```
p = vec2D(1,3)
p.x = 5 # objects are mutable
```

- In some languages, you can specify the access modifers for the attributes so that outsiders cannot access the objects attributes
- For Python, there are no access modifiers. To prevent outsiders from accessing attributes, use ***name mangling*** ...

Protecting Attributes

- Attributes with names beginning with __ are mangled

```
class vec2D:
    def __init__(self, x, y):
        self.__x = x
        self.__y = y
    def get_x(self): return self.__x
    def get_y(self): return self.__y
    def set_x(self, x): self.__x = x
    def set_y(self, y): self.__y = y
p = vec2D(1,3)
p.__x = 5
```

Within the class names are not mangled

Assignment

- WARNING: Try

```
p = vec2D(3,1)
q = p
print(id(p), id(q))
```

- What's the picture of the memory again?
- Once again:
  - = means q reference the same object as p
  - = does ***not*** mean q has a copy of p's data

Shallow and Deep Copy

- If you really want to copy the object a variable is pointing to to another, do the following:

```
import copy
p = vec2D(3,1)
q = copy.deepcopy(p)
print(id(p), p.get_x(), p.get_y())
print(id(q), q.get_x(), q.get_y())
```

Comparison

- WARNING:

```
p = vec2D(3,1)
r = vec2D(3,1)
print(p == r)
```

- What does == do? Now try ...

```
class vec2D:
    def __init__(self, x, y):
        self.__x = x
        self.__y = y
    def get_x(self): return self.__x
    def get_y(self): return self.__y
    def set_x(self, x): self.__x = x
    def set_y(self, y): self.__y = y
    def equals(self, q):
        return self.__x==q.__x and self.__y==q.__y
```

More Methods

```
class vec2D:
    def __init__(self, x, y):
        self.__x = x
        self.__y = y
    def get_x(self): return self.__x
    ...
    def equals(self, q): return self.__x==q.x and self.__y==q.y
    def add(self, q):
        s = vec2D(self.__x,self.__y)
        s.__x += q.__x
        s.__y += q.__y
        return s
p,q = vec2D(1,3), vec2D(5,2)
r = p.add(q) # EXPLAIN!!!
print(r.get_x(), r.get_y())
```

__repr__

- Are you sick of typing

```
print(p.get_x(), p.get_y())
```

You can't do `print` p, because you get

```
<__main__.vec2D instance at 0x00AA5FA8>
```

- Add this method:

```
class vec2D
    ...
    def __str__( self ):
        return "(%s,%s)" % (self.__x, self.__y)

p = vec2D(1,2)
print("p =", p) # EXPLAIN!!!
```

+

- I don't like "`r = p.add(q)`". So change the add method:

```
class vec2D:
    ...
    def __add__(self, q):
        s = vec2D(self.__x,self.__y)
        s.__x += q.__x
        s.__y += q.__y
        return s
    ...
p,q = vec2D(1,2), vec2D(5,2)
r = p + q # EXPLAIN!!!
print(r)
```

mult

- Exercise: Add a method `mult` so that

```
p = vec2D(2,3)
q = p.mult(2)
print(q)
```

- Gives `(4,6)`

*

- But I don't really like to type “`q = p.mult(2)`”. I prefer “`q = p*2`”.
- Change the name of your `mult` method to `__mul__`
- Now test your class by running:

```
p = vec2D(2,3)
q = p*2 # EXPLAIN!!!
print(q)
```

*

- Now try:

```
p = vec2D(2,3)
q = 2*p # BAD!!!
print(q)
```

- EXPLAIN!!! Now add `__rmul__` method:

```
class vec2D:
    ...
    def __rmul__(self, c):
        return vec2D(c*self.__x, c*self.__y)
    ...
```

and try the above code again. EXPLAIN!!!

*

- But __rmul__ in really just __mul__ with arguments reverse. So this works too:

```
class vec2D:
    ...
    def __rmul__(self, c):
        return self * c
    ...
```

Reminders

- Make sure you read this set of notes carefully and observer all the syntax
- If `x` is a `C`-object where `C` is a class, then

  ```
  x.f(y,z)
  ```

  is translated to

  ```
  C.f(x,y,z)
  ```

  and if in class `C` you have a method:

  ```
  def f(self,y,z): pass
  ```

  then `self` refers to the same object `x` is refering to when `f` executes.

Reminders

- The class is a scope (like namespace). So if you put your vec2D class in a `vec2D.py` file, then this is how you use it in another program file:

```
import vec2D
p = vec2D.vec2D(1,3)
```

Refers to the ***module***

Refers to the ***class*** in the module

Property

You can simplify calling get_x, set_x:

```
class vec2D:
    def __init__(self, x, y):
        self.__x = x
        self.__y = y
    def get_x(self): return self.__x
    def set_x(self, x): self.__x = x
    x = property(get_x, set_x)
    # etc.
```

After this you can do

```
p = vec2D(2, 3)
p.x = 5  # same as p.set_x(5)
a = p.x  # same as a = p.get_x()
```

Resource

- Make sure you read your C++ book on classes and objects. Most of the concepts are the same.