News

- Programming Assignment #6 is up, due next Monday
- Programming Assignment #7 on Prolog will be up soon and due at the end of the quarter

Python tries to avoid “overwrites”

Assignment Revisited

\[ x = e \]

1. Compute object corresponding to \( e \)
2. Change the name “\( x \)” to refer to object in the current namespace (added if missing)
What happens?

```python
>>> x = 10
def g():
    x = x + 1
    print x
>>> g()
>>> x
```

What happened?

- You may think it should print 11, and leave the global unchanged... but here is what really happens
- Since `x` is assigned in the function, it treats `x` as a local, and so it adds a binding for it.
- But then it looks up `x` for evaluating `x+1`, and it can’t find a local `x` ) ERROR

What happens?

```python
>>> x = 10
def f():
    x = x + 1
    print x
f()
```

What happened?

• You may think it should print 11, and leave the global unchanged... but here is what really happens

What happens?

```python
>>> x = [10]
def g():
    x[0] = “abc”
    print x
>>> g()
>>> x
```

What happened?

• You may think it should print 11, and leave the global unchanged... but here is what really happens
- Since `x` is assigned in the function, it treats `x` as a local, and so it adds a binding for it.
- But then it looks up `x` for evaluating `x+1`, and it can’t find a local `x` ) ERROR
What happens?

```python
>>> x = [10]
>>> def g():
...   x[0] = "abc"
...   print x
... g()
>>> x
['abc']
```

difference between assignment and mutation!

What happens?

```python
>>> x = [10]
>>> def h(z):
...   return (y+x[0]+z)
... return h

>>> foo = f(5)
>>> foo
<function object>
>>> foo(100)
>>> 115
>>> foo1 = f(-5)
>>> foo1(100)
105
```

Rest of the quarter

- Today: Classes and Inheritance in Python
- Th 2/26: Decorators
- Tu 3/3: Prolog, part 1
- Th 3/5: Prolog, part 2
- Tu 3/10: Conclusion
- Th 3/12: class canceled (visit day)
Let’s talk about objects

- Namespaces == object
- What ways have we seen of creating a namespace?
- Go to code

But this sucks

- Why is this not good enough for object oriented programming?

But this sucks

- Why is this not good enough for object oriented programming?

Class-based model

- Have classes that describe the format of objects
- Create objects by stating the class of the object to be created.
- The created object is called an instance of the class
Class-based model

- In a class based model, the class is sometimes an object too (as is the case in Python)

- Q: what is the class of the class object?

What’s the alternative?

- Suppose we didn’t have classes

- How would one survive?

Prototype-based models

- Just have objects
  - Create a new object by cloning another one
  - Add/update fields later

- Benefits:
  - Simplifies the definition of the language
  - Avoids meta-class problem

- Drawbacks:
  - Don’t have classes for static typing
  - Some find the model harder to grok

- Python has hints of a prototype-based language. Go back to code
Methods

Next: constructors

• Go back to code

Inheritance

• Key concept of OO languages

• Someone tell me what inheritance is?
### Inheritance

- Key concept of OO languages

- Someone tell me what inheritance is?
  - “is-a” relationship

- Examples?

### Overriding

- Super-class method can be overwritten in sub-class
- Polymorphism
  - external clients write code that handles different kinds of objects in **uniform** way
  - don’t care about implementation details: as long as the object knows to draw itself, that’s good enough (**see screensaver on webpage**)

### Examples of inheritance

### Polymorphism, continued

- Super-class can have methods that are not overridden, but that work differently for different sub-classes

- For example: super-class method functionality changes because the super-class calls a method that gets overwritten in the sub-class
Simple example

```python
class Shape:
    def draw(self, screen):
        # some python code here
    def erase(self, screen):
        screen.setcolor("white")
        self.draw(screen)
        screen.setcolor("black")

class Rec(Shape):
    def draw(self, screen):
        # some python code here

class Oval(Shape):
    def draw(self, screen):
        # some python code here
```

Joon’s Question

How can “Oval”’s draw method call the superclass’s draw method?

Stepping away from Python

- What are the fundamental issues with inheritance?

Stepping away from Python

- What are the fundamental issues with inheritance?
- Dispatch mechanism?
  - most compilers use v-tables
  - more complicated with multi-methods
Stepping away from Python

• What are the fundamental issues with inheritance?

• Overloading vs. Overriding?
  - what’s the difference?

Rectangle and Square

```python
class Rectangle:
    length = 0
    width = 0
def area(this):
    return this.length * this.width
```

```python
class Square:
    length = 0
def area(this):
    return this.length * this.length
```

• Which should be a sub-class of which?

• Answer is not clear…

Stepping away from Python

• What are the fundamental issues with inheritance?

• How to decide on the inheritance graph?
  - not always obvious, see next example
Option 1: Rectangle is-a Square

```python
class Square:
    length = 0
def area(this):
        return this.length * this.length

class Rectangle(Square):
    width = 0
def area(this):
        return this.length * this.width
```

Option 2: Square is-a Rectangle

```python
class Rectangle:
    length = 0
    width = 0
def area(this):
        return this.length * this.width

class Square(Rectangle):
    __init__(self, len):
        self.length = len
        self.width = len
```

But, does it matter? Performance is a tricky matter. Often better to implement first, then use profiler to find where bottlenecks are.
Option 3:

```
class Shape:
    ...

class Rectangle(Shape):
    length = 0
    width = 0
    def area(this):
        return this.length * this.width

class Square(Shape):
    length = 0
    def area(this):
        return this.length * this.length
```

- Store only what is needed (one field for square)
- Does not follow “isa” relationship from math (rectangle is not a square...)
- Have to write two area methods

Complex numbers

```
class Real:
    RealPart = 0

class Complex:
    RealPart = 0
    ComplexPart = 0
```

The same exact options present themselves here, with the same tradeoffs!

Summary of (single) inheritance

- Inheritance is a powerful mechanism
- For the programmer,
difficulty = defining inheritance diagram
- For the language implementer,
difficulty = fast dynamic dispatch
Next time

- Multiple inheritance

- Decorators