CS 240
Fall 2014

Mike Lam, Professor

Object-Oriented Python

Today

- Documentation tips
- Python objects
- Python inheritance
- Python type system

Code Documentation

- Avoid one-letter names ("a", "b", etc.)
  - Exception: loop indices (but match type: “i” vs. “ch
- Short but descriptive names
  - If it's not a list, don't call it “name_list”
- Check for duplicate in-scope variables
  - Nested loops, conditional bodies

Code Documentation

- In-line comments
  - Explain tricky code
  - Don't just echo the code
  - If you forget what it's doing after 48 hours, you won't understand it six months later
- Docstrings
  - String literal that appears as the first (indented) statement in a module, class, or function
  - Uses """ delimiters by convention

Why Objects?

Why Objects?

- Reusability
  - Don't re-invent the wheel
- Modularity
  - Easier to manage and test
- Abstraction
  - Cleaner designs

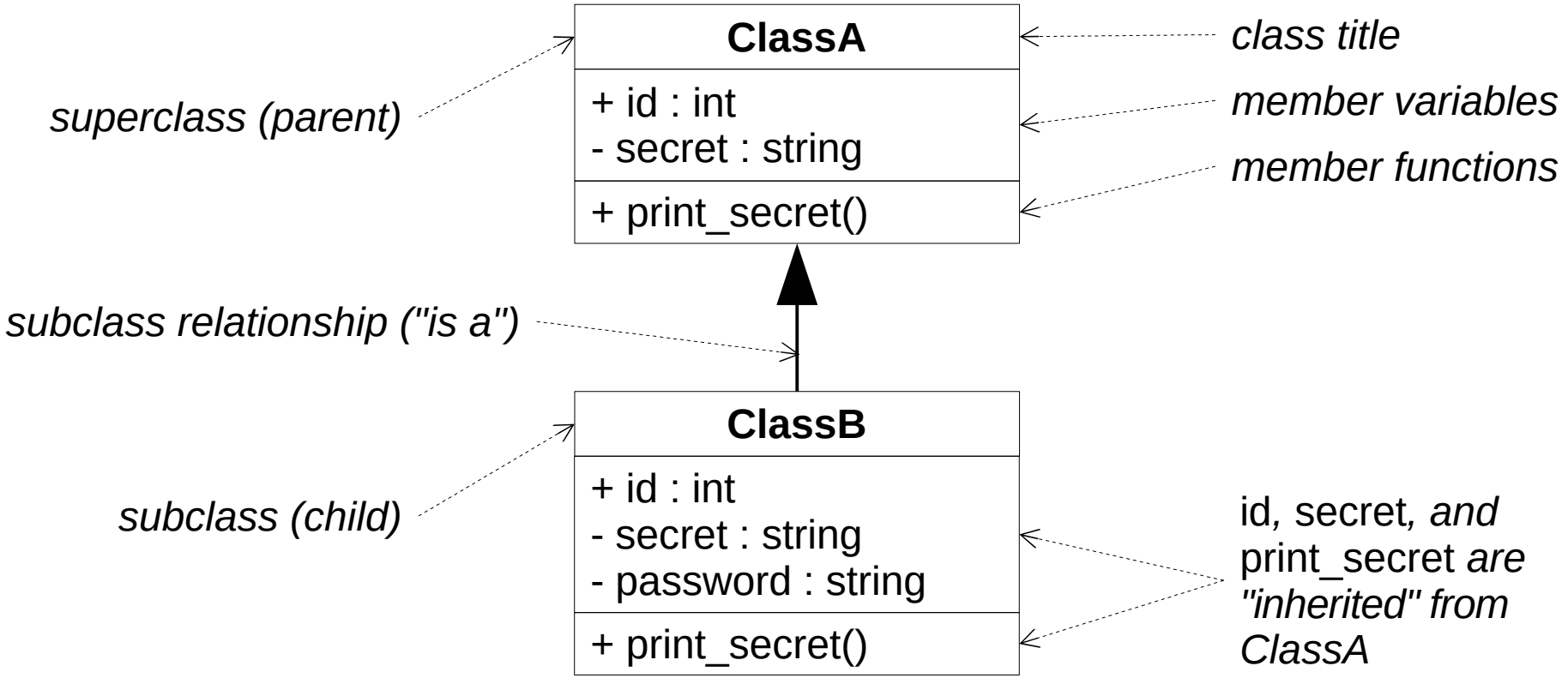
Object Components

Object Components

- Fields
  - Member variables
  - Public vs. private (convention-only in Python)
    - `obj.xvs.obj._x`
- Behaviors
  - Member methods/functions
  - Explicit "self" in Python

UML

- Unified Modeling Language



Python Classes

```
class Polygon():
    """ Represents an n-sided Polygon in a 2d plane """

    def __init__(self, x, y, sides, size):
        self._x = x
        self._y = y
        self._sides = sides
        self._size = size

    def __str__(self):
        return str(self._x) + ", " + str(self._y)

    def __eq__(self, rhs):
        return (self._x == rhs._x and self._y == rhs._y)


tri   = Polygon(0, 0, 3, 10)
penta = Polygon(5, 5, 5, 10)
hexa  = Polygon(5, 5, 6, 15)

print(str(tri))                         # same as print(tri.__str__())

print(penta == hexa)
```

Python Classes

- All methods take "self" parameter
  - `x.foo()` means `Class.foo(x)`
- `__init__` is the constructor
- `__str__` is called when the user says `str(x)`
- `__eq__` is called when the user says `x == y`
  - "`is`" is different (reference vs. value equality)
- All members are public
  - Convention: use "_" prefix to mark private members

Python Inheritance

```
class Polygon():
    def __init__(self, x, y, sides, size):
        self.x = x
        self.y = y
        self.sides = sides
        self.size = size

    def __str__(self):
        return str(self._x) + ", " + str(self._y)

class Square(Polygon):
    def __init__(self, x, y, size):
        super().__init__(x, y, 4, size)


sq = Square(5, 10, 50)
print(str(sq))                      # Square inherits __str__()
                                    # from Polygon
```

Python Typing

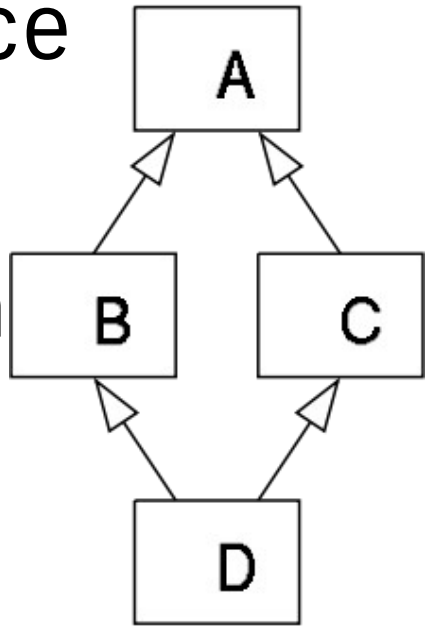
- Python is dynamically typed
  - No types explicitly declared in code
  - All variables are references
    - All values are objects
  - Objects do have a type at runtime!
    - Assigned during initialization (“=” operator)
    - Call `type(x)` to see x's type
  - Python uses “dynamic dispatch”
    - Interpreter checks for members at run time

“Duck” Typing

- "If it walks like a duck and talks like a duck...
  - ... treat it like a duck."

Multiple Inheritance

- Python allows multiple inheritance
- "Diamond problem"
  - C3 algorithm for method resolution



- We won't use multiple inheritance in this class

Abstract Base Classes

- Non-instantiable parent class
- Marks methods that all subclasses (children) should impleme
- In Python, use the `abc` module:

```
from abc import ABCMeta, abstractmethod

class MyClass(metaclass=ABCMeta):

    @abstractmethod
    def do_something():       # subclasses
        pass                  # must implement

my_obj = MyClass()            # TypeError!
```

Python Classes

- Example

Reminders

- Homework 1 due on Friday @ 14:30 (2:30 pm
  - Submit on Canvas
- PA1 posted
  - Due September 17 @ 12:00 (noon)
- Lab on Friday
  - Shape hierarchy
  - Time to work on previous labs