Classes and Objects

Introduction

Copyright © Software Carpentry 2010
This work is licensed under the Creative Commons Attribution License
See http://software-carpentry.org/license.html for more information.
Computer science is the study of algorithms
Computer science is the study of algorithms

Computer *programming* is about creating and composing *abstractions*
Computer science is the study of algorithms

Computer *programming* is about creating and composing *abstractions* to hide the details.
Computer science is the study of algorithms.

Computer *programming* is about creating and composing *abstractions* to hide the details and make one thing act like another.
Computer science is the study of algorithms

Computer *programming* is about creating and composing *abstractions* to hide the details and make one thing act like another.

Functions turn many steps into one (logical) step.
Computer science is the study of algorithms

Computer *programming* is about creating and composing *abstractions* to hide the details and make one thing act like another.

Functions turn many steps into one (logical) step.

Libraries group functions to make them manageable.
Computer science is the study of algorithms

Computer *programming* is about creating and composing *abstractions* to hide the details and make one thing act like another.

Functions turn many steps into one (logical) step.

Libraries group functions to make them manageable.

Classes and objects combine functions and data.
Computer science is the study of algorithms

Computer *programming* is about creating and composing *abstractions*.

- Hide the details
- Make one thing act like another

Functions turn many steps into one (logical) step.

Libraries group functions to make them manageable.

Classes and objects combine functions and data.

And, if used properly, do much more as well.
Simple simulation of aquarium containing
Simple simulation of aquarium containing plants
Simple simulation of aquarium containing

plants

snails
Simple simulation of aquarium containing plants, snails, and fish.
Simple simulation of aquarium containing plants, snails, and fish. Plants don't move and photosynthesize.
Simple simulation of aquarium containing

plants

snails

fish

don't move
crawl in 2D

photosynthesize

scavenge
Simple simulation of aquarium containing

plants  snails  fish

  don't move  crawl in 2D  swim in 3D

photosynthesize  scavenge  hunt
Simple simulation of aquarium containing

- plants
  - don't move
- snails
  - crawl in 2D
- fish
  - swim in 3D
- photosynthesize
- scavenge
- hunt

Algorithm is simple
Simple simulation of aquarium containing plants, snails, and fish. Plants don't move, snails crawl in 2D, and fish swim in 3D. Plants photosynthesize, snails scavenge, and fish hunt.

Algorithm is simple:

```python
for t in range(timesteps):
    move(world, everything)
    eat(world, everything)
    show(world,)
```
Simple simulation of aquarium containing plants, snails, and fish. Plants don't move, snails crawl in 2D, and fish swim in 3D. Plants photosynthesize, snails scavenge, and fish hunt.

Algorithm is simple:

```python
for t in range(timesteps):
    move(world, everything)
    eat(world, everything)
    show(world,)
```

Program is more complicated.
def move(world, everything):
    for thing in everything:
        if thing[0] == 'plant':
            pass  # plants don't move
        elif thing[0] == 'snail':
            move_snail(snail)
        elif thing[0] == 'fish':
            move_fish(fish)
def move(world, everything):
    for thing in everything:
        if thing[0] == 'plant':
            pass  # plants don't move
        elif thing[0] == 'snail':
            move_snail(snail)
        elif thing[0] == 'fish':
            move_fish(fish)

So far, so good
def eat(world, everything):
    for thing in everything:
        if thing[0] == 'plant':
            photosynthesize(world, plant)
        elif thing[0] == 'snail':
            scavenge(world, snail)
        elif thing[0] == 'fish':
            prey = hunt(world, everything, thing)
            if prey != None:
                devour(world, everything, thing, prey)
def eat(world, everything):
    for thing in everything:
        if thing[0] == 'plant':
            photosynthesize(world, plant)
        elif thing[0] == 'snail':
            scavenge(world, snail)
        elif thing[0] == 'fish':
            prey = hunt(world, everything, thing)
            if prey != None:
                devour(world, everything, thing, prey)

Hmm...
def show(world, everything):
    show_world(world)
    for thing in everything:
        if thing[0] == 'plant':
            show_plant(plant)
        elif thing[0] == 'snail':
            show_snail(snail)
        elif thing[0] == 'fish':
            show_fish_fish
def show(world, everything):
    show_world(world)
    for thing in everything:
        if thing[0] == 'plant':
            show_plant(plant)
        elif thing[0] == 'snail':
            show_snail(snail)
        elif thing[0] == 'fish':
            show_fish_fish

This is starting to look familiaré
Pessimist: code that's repeated in two or more places will eventually be wrong in at least one
Pessimist: code that's repeated in two or more places will eventually be wrong in at least one. To add starfish, we have to modify three functions.
Pessimist: code that's repeated in two or more places will eventually be wrong in at least one. To add starfish, we have to modify three functions. 

\[ \text{Remember to} \]
Pessimist: code that's repeated in two or more places will eventually be wrong in at least one.

To add starfish, we have to modify three functions remember to

What about fish that eat plants? Or scavenge?
Pessimist: code that's repeated in two or more places will eventually be wrong in at least one
To add starfish, we have to modify three functions remember to
What about fish that eat plants? Or scavenge?
Optimist: every pattern in a program is an opportunity to shorten that program
Wouldn't this be simpler?
Wouldn't this be simpler?

```python
for thing in everything:
    thing.move()
prey = thing.eat(everything)
if prey:
    thing.devour(prey)
    everything.remove(prey)
```
Wouldn't this be simpler?

```python
for thing in everything:
    thing.move()
prey = thing.eat(everything)
if prey:
    thing.devour(prey)
    everything.remove(prey)
```

Easier to understand (after some practice)
Wouldn't this be simpler?

```python
for thing in everything:
    thing.move()
    prey = thing.eat(everything)
    if prey:
        thing.devour(prey)
        everything.remove(prey)
```

Easier to understand (after some practice)

*Much* easier to add new kinds of things
Nothing is free
Nothing is free

Simple programs become slightly more complex
Nothing is free
Simple programs become slightly more complex
And too much abstraction creates as big a mental burden as too little
Nothing is free
Simple programs become slightly more complex
And too much abstraction creates as big a mental burden as too little

Degree of Abstraction

Mental Effort Required

Putting steps together to get big picture
Nothing is free
Simple programs become slightly more complex
And too much abstraction creates as big a mental burden as too little