Computer Science 134C
Introduction to Computer Science, in Python
Lecture #5 (The Art of Finding and Hiding Details)
September 17, 2018

Keywords
abstraction, assertions, encapsulation, equivalence, identity, interface/implementation, pre- & postconditions, public/private, refactoring

We start to think about abstraction and design.

1. HW 1 due, now. Lab 2 today and tomorrow — compute the age of the moon.
2. Questions?
3. From before: Booleans—basic operations—combining values—interaction with non-booleans—identity *vs.* equivalence testing and `None`.
4. From before: Full details on `if`—forms—nesting–`pass`.
5. From before: The `while` loop—`break`—`continue`.
6. From before: what are long orbits of `syr`, the $3n + 1$ function?

⋆ ⋆ ⋆

7. Abstraction: the result of "seeing only the big picture," by hiding the details. Important related ideas:
   (a) The *interface* is the "front-facing," *public* side of the system.
   (b) The *implementation* is the hidden, *private* details of how the interface is actually accomplished.
   (c) *Encapsulation* is typically used to confine the implementation details and ensure their privacy.
   (d) The *preconditions* are assumed to be true if we make use of the interface.
   These are tested by *assertions*.
   (e) Provided the preconditions hold, the *postcondition* is what will be true once we use the system.
8. Unfortunately, we sometimes fail to appreciate the correct level for abstraction.
   Like any craft, the road to improvement sometimes involves repeated attempts at design.
   (a) The process of reducing the preconditions is called *generalization*.
   (b) Because a system is often made up of several components that must work together, the process of reworking the design in a *consistent* manner is called *refactoring*.
9. General truths.
   (a) Design is hard and takes patience.
   (b) Poor interface design leads to burdensome, *legacy* client code.
   (c) Significant refactoring is an indication of flaws in the original design.

⋆