Design by Contract

CSC148, Introduction to Computer Science
Fall 2016
Diane Horton and David Liu
Docstrings

• Our function/method design recipe yields both the code and a docstring.

• Example:
  
def decreases_at(L):
      """Return the index of the first integer in L that is less than its predecessor."
      
      @type L: list(int)
      @rtype: int
      >>> decreases_at([3, 6, 9, 12, 2, 1, 8, 5])
      >>> 4
      """
Good docstrings

• Principles:
  – Describe **what** the method does, very precisely, in every circumstance.
  – Mention every parameter by name.
  – Do not discuss **how** the method works
    E.g., Do not discuss local variables, helper methods, or algorithms.

• Can think of a docstring as specifying a contract.
Contracts in real life

• A binding agreement between two parties.

• Example:
  Provided that the client arrives 2 hours prior to departure, having paid the full fair in advance, and with luggage that is within set limits, the airline will fly the client and luggage to their destination.

• What if
  – Arrives 3 hours prior to departure?
  – Arrives 10 minutes prior to departure?
  – Brings luggage that exceeds the limits?
What the contract means

• If the client meets the conditions:
  The promised actions will occur.
• If not:
  No guarantees!
Contracts in programming

• A function or method’s pre-condition is a boolean expression that describes
  – What must be true at the beginning of a call.
  – That is, restrictions on the values of the arguments.

• A function or method’s post-condition is a boolean expression that describes
  – what will be true at the end of the call.

• If the pre-conditions are met:
  1. The function will halt without crashing, and
  2. Will satisfy the post-condition.

• If not: No guarantees!
Design by contract

• **Design by contract** is a way of thinking about programming.

• Each function (or method) provides a service.

• Its service is specified as a contract by the docstring, including:
  
  – What it expects (the precondition)
  – What it guarantees (the postcondition)
A little history

- Bertrand Meyer invented the metaphor in 1986.
- Part of his work on the Eiffel programming language.
- An `assert` statement lets you make the conditions executable.
- Eiffel, and many languages since, has assertions.
- You saw them in Python when you used `unittest`. 
Uses for a function/method contract

• Guides you in writing the body.
• Defines an API for client code.
• Aids in debugging.
• Aids in code maintenance.
  – If you improve the implementation, you know what must still be guaranteed.
• Helps you argue that the code is correct.
  – Defines what “correct” means.
  – Is the basis for a proof of correctness.
Checking the pre- and post-conditions

• We can use asserts to check the pre- and post-conditions.

• [Example from Assignment 2]

• Type contracts are a particular category of pre-/post-condition.
  – In many languages, the compiler checks these automatically
  – Then you don’t need to add asserts to check them yourself
Using asserts within a method

• Code often has key points where certain properties should be true.

• E.g., we’ve seen code with this structure:
  Read and process data files
  Read and process configuration
  # Everything is set up
  Perform the algorithm

• We can use asserts to express these properties.
Perhaps break down the code

• You may find you have several such moments:
  Steps
  More steps
  *Assert some things*
  More steps
  Even more
  *Assert some other things*

• This may suggest chopping into helpers.

• Then the asserts become post and pre-conditions of the helpers.
Loop invariants

• Design By Contract is also helpful with loops.
• We specify what will be true on each iteration.
• A loop invariant is a boolean condition that is true at the beginning and end of each iteration of the loop.

• [Examples from iterative searching]
How to use loop invariants

Define conditions and code in this order:

1. Outcome: What is the code supposed to do?
   - These are postconditions of the loop.

2. Given: What has to be true in order for this to happen?
   - These are preconditions of the loop.

3. Loop invariant: What change would get us closer to the goal on each iteration?
   - This suggests the code for the loop body!
Useful?

• For simple loops, we don’t bother writing down loop invariants.

• But for tricky ones, articulating the invariant helps us
  – Write the code.
  – Be sure, or even prove, that the algorithm is correct.
  – Debug the code.
    Stop on that line and check if the L.I. is true!

• And it helps others to understand the code.

• Loop invariants are a powerful concept.
Consider binary search

• Bentley, 2000:
  “I’ve assigned this problem in courses for professional programmers. The students had a couple of hours to convert [a description of binary search] into a program in the language of their choice ... **Only about ten percent of professional programmers** were able to get this small program right.”

• Knuth, 1973:
  – Binary search first published: 1946.
Checking the loop invariants

- We can declare loop invariants in a comment.
- We can also check them using assert.
- Any down side to checking them?
Best of both worlds

• In many languages, you can run a program with asserts disabled.
• Python supports this.
• Workflow:
  – During development use asserts to detect bugs.
  – In production disable them, if appropriate, for performance reasons.