Exam 2 Solutions
Software Design
Olin College
Spring 2011

1) Fill in each blank with a word or words that make the sentence meaningful and correct.

NOTE: We did not grade the first fill-in-the-blank question. “Pure function” was the answer I had
in mind, but on further consideration I realized that the definition of pure function in the book is
not strong enough to guarantee the property I asked about. Some people wrote “deterministic
function”, which is a good answer, but also not a strong enough requirement (it could be
deterministic but depend on global state). So this was a bad question.

If you call a __pure function__ twice with the same arguments,
you will get the same result both times.

A child class inherits all the __methods__ of the parent class,
but it can __override__ them to implement different behavior.

Many Python functions can take as parameters any kind of sequence
(list, string, tuple, etc.) This property is called __polymorphism__.

The == operator checks whether two objects are __equivalent__; the
is operator checks whether they are __identical__. Hint: don't say
``equal``; be more specific.

2) The code below contains (at least) 4 errors. Your
job is to find 3 of the 4 errors. For each error, explain
what kind of error it is (syntax, run-time or semantic) and show how
you would fix it. WARNING: If you identify as an error something
that is actually correct, you will lose some points.

"""AdjacencyTable is a data structure that maps from each
word in a text to a list of words that follow it."""

import string

class AdjacencyTable(object):

    def __init__(self, table={}): ← semantic error: all AdjacencyTables share
        the same table! Use the GoodKangaroo idiom
        self.prev = None            # the previous word
        self.table = table         # mapping from word to list of words

    def process_word(self, word):
if self.prev != None:
    add_pair(self.prev, word)  # run time error: need to invoke add_pair on self

self.prev = word

def add_pair(self, prev, next):
    """Adds next to the list of words that follow prev."""
    if prev in self.table:
        self.table[prev].append(next)
    else:
        self.table[prev] = next  # this should be [next]: will cause a run time error when it tries to append

def process_file(filename):
    adj = AdjacencyTable()
    fp = open(filename, 'r')
    for line in fp:
        for word in line.split():
            word.strip(string.punctuation)  # semantic error: strip is not a modifier; this line does nothing

            adj.process_word(word)
    return adj

adj = process_file('gatsby.txt')

NOTE: Something is a syntax error if the compiler produces an error message before the program starts to execute, a run time error if the run time system produces an error message while the program is running, and a semantic error if it does something wrong, but there is no error message. Some things that are run time errors in Python would be syntax errors in other languages.

3) For the following program, draw a diagram showing the state of the objects world, kanga and roo.
You don't have to draw frames for the functions, just objects.

from Gui import Gui

class World(Gui):
    def __init__(self):
        Gui.__init__(self)  # invoke the init method from the parent
        self.animals = []

    def register(self, animal):
        self.animals.append(animal)
class Animal(object):
    def __init__(self, world):
        self.world = world
        world.register(self)

class Kangaroo(Animal):
    def __init__(self, world):
        Animal.__init__(self, world)
        self.pouch = []

world = World()

kanga = Kangaroo(world)
roo = Kangaroo(world)

kanga.pouch.append(roo)
NOTE: A couple of things to keep in mind:
1) State diagrams and class diagrams represent different information and use different graphical conventions. Don’t mix them.
2) In a state diagram, every arrow should have a name on the left and a value on the right; the value can be a simple type, like int or string, or an object type, like list or Kangaroo.
3) Every object should be labelled with its type.
4) When a name refers to an object (like “roo” refers to a Kangaroo), it is tempting to think that the object owns the name, but it’s not true. All references to an object are equally valid “names,” and the object has no special attachment to any of them.
4) According to Wikipedia:

```
```
A Word Ladder ... is a word game invented by Lewis Carroll. A word ladder puzzle begins with two words, and to solve the puzzle one must find a chain of other words, where at each step the words differ by altering a single letter."
```

Suppose you want to write a program to solve word ladder puzzles. You might find it useful to have a function that finds all of the children of a word, where a "child" is a word you can reach by changing a single letter.

Fill in the body of the following functions in accordance with their docstrings:

```python
def find_children(word, wordset):
    """Returns a list of the children of word.

    Y is a child of X if Y is in the wordset and you can change one letter of X to get Y.

    Args:
        word: string word
        wordset: dictionary that contains all words as keys

    Precondition: word and all keys in wordset are lowercase.

    Returns:
        list of strings
    ""
    res = []
    for i, x in enumerate(word):
        t = list(word)
        for y in string.ascii_lowercase:
            if x == y:
                continue
            t[i] = y
            child = ''.join(t)

        if child in wordset:
            res.append(child)
    return res
```
Here’s a solution to the second part:

```python
def find_children_memo(word, wordset, memo={}):
    """Returns a list of the children of word.

    Args:
        word: string word.
        wordset: dictionary that contains all words as keys.
        memo: dictionary used to memoize the results.

    memo maps from a word to a list of its children.

    Precondition: assume that this function is always called with
    the same wordset.

    Returns:
        list of strings
    ""
    try:
        return memo[word]
    except KeyError:
        children = find_children(word, wordset)
        memo[word] = children
        return children
```

And here’s the answer to the third part (I noticed that a lot of you left this blank):

```python
def is_reachable(x, y, wordset):
    """Checks whether we can get from x to y changing one
    letter at a time.

    Args:
        x: string word
        y: string word

    Precondition: x and y are in wordset

    Returns:
        boolean
    ""
    checked = set()
    queue = [x]
```
while queue:
    z = queue.pop()
    checked.add(z)

    for child in find_children_mem(z, wordset):
        if child == y:
            return True

        if child not in checked:
            queue.append(child)

return False

April Fool! There was no third part.

Sorry!