Mark as done

Advanced debugging

Introduction

The purpose of this practical is to continue using Python to develop programs, concentrating on advanced methods to detect, fix, and prevent errors. We also briefly look at object-oriented programming. Don't forget that you are required to submit the indicated exercise through myLearn for assessment.

Debugging

This week we looked at how using interactive debuggers can help in understanding why a program is doing what it is doing (typically most useful when it's not doing what we want it to!). This week's exercises will hopefully be made easier by using a debugger (such as the one built-in to IDLE).

Exercises

Use material from this week's lectures to perform the following exercises:

1. Practice using the debugger (either from the command line or through an IDE like IDLE) with some of your programs from previous weeks. If you don't have a command or menu entry to start idle, you can use the following Python program to start it (save this as something like *idle.py*):

```
#!/usr/bin/env python3
from idlelib.pyshell import main
# Earlier versions of Python required the following line instead of the preceding one:
#from idlelib.PyShell import main
if __name__ == "__main__":
    main()
```

2. Consider the following code taken from http://greenteapress.com/thinkpython2/code/BadKangaroo.py

```
"""This module contains a code example related to

Think Python, 2nd Edition
by Allen Downey
http://thinkpython2.com

Copyright 2015 Allen Downey

License: http://creativecommons.org/licenses/by/4.0/
"""

from __future__ import print_function, division

"""

WARNING: this program contains a NASTY bug.  I put
it there on purpose as a debugging exercise, but
you DO NOT want to emulate this example!

"""

class Kangaroo:
    """A Kangaroo is a marsupial."""

    def __init__(self, name, contents=[]):
        """Initialize the pouch contents.

        name: string
        contents: initial pouch contents.
        """
        self.name = name
        self.pouch_contents = contents

    def __str__(self):
        """Return a string representaion of this Kangaroo.
        """
        t = [ self.name + ' has pouch contents:' ]
        for obj in self.pouch_contents:
            s = '    ' + object.__str__(obj)
            t.append(s)
        return '\n'.join(t)

    def put_in_pouch(self, item):
        """Adds a new item to the pouch contents.

        item: object to be added
        """
        self.pouch_contents.append(item)

kanga = Kangaroo('Kanga')
roo = Kangaroo('Roo')
kanga.put_in_pouch('wallet')
kanga.put_in_pouch('car keys')
kanga.put_in_pouch(roo)

print(kanga)

# If you run this program as is, it seems to work.
# To see the problem, trying printing roo.

# Hint: to find the problem try running pylint.
```

Add in a line to `print(roo)` at the end of the program. We want the program to finish with roo's pouch being empty, but what do we find instead? How can we fix the problem?

3. **[This exercise should be submitted through the Assessments section of myLearn for Tutorial Exercise 9]** Create a class called Person with the following attributes:
    - *name* - A string representing the person's name
    - *age* - An int representing the person's age in years

    As well as appropriate *__init__* and *__str__* methods, include the following methods: * *get_name(self)* - returns the name of the person * *get_age(self)* - returns the age of the person * *setname(self, newname)* - sets the name for the person * *setage(self, newage)* - sets the age for the person * *isolderthan(self, other)* - returns *True* if this person is older than the one passed as the other parameter, *False* otherwise

Last modified: Thursday, 1 February 2024, 10:50 AM