**IT FDN 100A: Foundations of Programming: Python**

**Instructor: Randal Root**

**Student: Morgan Lang**

**July 12, 2016**

# Assignment 4

## Assignment objectives:

1. Create a program that asks the user for the name of a household item, then asks for its estimated value. Store both pieces of data in a tuple, then move the data from the tuple to a text file called HomeInventory.txt
2. Test the program.
3. Paste the resultant code into a Word document in which you describe your learning experience.
4. Submit the Word document via Canvas.

## Overview

I had encountered tuples in Python before, but I never understood them. Now I get it: they’re just lists, but are unique in that they are immutable. Most of my experience with lists comes from using arrays in JavaScript, so I understood the concept and syntax of interacting with index points in a list. However, I had never really done anything with indexed list items other than write and read them, so it was good to have a chance to perform some formatting operations on them.

I have commented the code to explain individual steps.

## The program

About 85% of this code has been copied from our previous assignment. The main changes applied to the program were:

1. The core functionality of the program has been updated to use a tuple to store user input.
2. Comments have been expanded for the peer-review process.
3. A few more user affordances have been provided.

I originally started by jumping right into asking the user for values to populate the file, but I wanted to add some logic to (a) create the inventory file if it didn’t exist and (b) display any extant content of the inventory file to the user. My workaround was to open the file in **append** mode, then close it again, then open it again in **read** mode, then close it again, and finally open it in **append** mode again. This is probably not the most elegant way to accomplish my goal, but I can only use the tools of which I’m aware.

The keys to completing this assignment were:

1. Understanding the syntax involved in defining file operations as variables. Once I understood that a file operation (such as opening a file) could be defined and used just like any other variable, things became easier.
2. Understanding that tuples can contain lists of almost anything, including user inputs. This knowledge allowed me to place two user prompts within a tuple.

As usual, I have tried to use UCD (User-Centered Design) principles in the program design, using pauses to pace the program and providing feedback to the user whenever appropriate.

**HomeInventory-week04.py:**

"""  
Program name: HomeInventory-week04.py  
Program function: to create an inventory file, display its contents, and allow user input to be added to it.  
User inputs are initially added to a tuple, then the user inputs are sliced out of the tuple, reformatted, and  
written to the inventory file when the user chooses to stop adding items.  
Author: Morgan Lang  
Date modified: 07/12/2016  
  
NOTE: The following directory must exist in order to run this program: C:\\\_Pythonclass  
"""  
  
**import** time  
  
# Splash / welcome screen. Yes, it's goofy. I don't care.  
print(  
 '''  
 \_\_\_ \_ \_ \_ \_ \_\_\_  
 |\_ \_|\_ \_\_\_ \_\_\_\_\_ \_ \_| |\_ \_\_\_ \_ \_ \_ \_ /\_\ \_\_| |\_\_| |\_\_\_ \_ \_ \_\_\_ / \_ \ \_ \_ \_ \_ \_\_\_ \_ \_  
 | || ' \ V / -\_) ' \ \_/ \_ \ '\_| || | / \_ \/ \_` / \_` / -\_) '\_|\_\_\_| (\_) | ' \| ' \/ -\_) '\_|  
 |\_\_\_|\_||\_\\_/\\_\_\_|\_||\_\\_\_\\_\_\_/\_| \\_, | /\_/ \\_\\_\_,\_\\_\_,\_\\_\_\_|\_| \\_\_\_/|\_||\_|\_||\_\\_\_\_|\_|  
 |\_\_/  
  
 '''  
)  
  
  
# First we'll need to create our inventory file if it doesn't exist. We'll do that by opening it in append mode.  
# If it doesn't exist, it will be created. There's probably a better way to do this.  
inventoryFileCreate = open("C:\\\_Pythonclass\\HomeInventory.txt", "a")  
print("\nInventory file located at C:\\\_Pythonclass\\HomeInventory.txt")  
time.sleep(1)  
# Closing the file so that inventoryFileRead can open it below and show its contents to the user  
inventoryFileCreate.close()  
  
# Now we'll display the current contents of the inventory by opening the target file in 'read' mode:  
inventoryFileRead = open("C:\\\_Pythonclass\\HomeInventory.txt", "r")  
print("\nCurrent inventory: \n\n", inventoryFileRead.read())  
# Closing the file so that inventoryFileWrite can open it below  
inventoryFileRead.close()  
  
# Now we'll open our inventory file in 'append mode' so we can actually do some work with it:  
inventoryFileWrite = open("C:\\\_Pythonclass\\HomeInventory.txt", "a")  
  
# Giving the user some context and basic information.  
print("Welcome to the Inventory Adder-Onner.\n"  
 "This program adds inventory items to the following file: C:\\\_Pythonclass\\HomeInventory.txt\n\n")  
  
# Pausing for dramatic effect. Users like it when computers slow down to their speed.  
time.sleep(1)  
  
# We want to use a while loop because we need to keep prompting for input until the user chooses to quit.  
**while True**:  
 # Prompting for an item description and value. Each user input becomes an element in a tuple:  
 userInput01 = ((input("Enter an item to add to the inventory file. \n"  
 "Use as much description as you think is necessary: ")),  
 (input("Enter the value of the item.\n"  
 "It isn't necessary to use a dollar sign: ")))  
  
 # Slicing and formatting the index points in our tuple.  
 # Adding spaces, a dash, and a dollar sign helps readability.  
 # We're also removing any user-supplied dollar signs from the 'value' data to avoid inconsistency.  
 formattedInput = userInput01[0] + " " + "-" + " " + "$" + userInput01[1].replace("$", "")  
  
 # Write the formatted output and add a new line for the next item.  
 inventoryFileWrite.write(formattedInput + "\n")  
  
 # It's always good to provide feedback and context so the user can understand what's happening.  
 print("You entered:", formattedInput)  
 time.sleep(.2)  
 print("This item will be added to the inventory upon exiting the program. \n")  
  
 # Providing the user with a way to exit or continue the program  
 userInput03 =(input("Press the 'enter' or 'return' key to continue adding items."  
 "\n Type 'save' to add the new inventory item(s) and quit this program. \n"))  
  
 # The exit condition. We'll convert userInput03 to lowercase to be sure we catch 'Save' and 'SAVE.'  
 # Note that we're fibbing to the user, since the data were already added at the inventoryFileWrite.write stage above  
 **if** userInput03.lower() == "save":  
 print("Adding item(s) to inventory file and preparing to exit...")  
 time.sleep(.5)  
 **break**# The user has initiated the exit condition and broken out of the while loop,  
# so we're outdenting and wrapping things up.  
# Closing the working inventory file:  
inventoryFileWrite.close()  
  
# Now we'll open and read the completed inventory file with the most recent addition(s).  
# This provides reassurance to the user (and to us!) that the items really have been added.  
inventoryFileRead = open("C:\\\_Pythonclass\\HomeInventory.txt", "r")  
print("\nCurrent inventory: \n\n", inventoryFileRead.read())  
# Closing the file again  
inventoryFileRead.close()  
print("Thanks for using the Inventory Adder-Onner. See you next time!")  
time.sleep(1)  
input("\nPress the Return or Enter key to close the program.")

## Miscellany:

Note that copying the ASCII art sometimes creates display problems, as seen above. The running code displays it correctly.