COLORADO SCHOOL OF MINES
ELECTRICAL ENGINEERING & COMPUTER SCIENCE DEPARTMENT

CSCI-410
Elements of Computing Systems
Spring 2014

PY-01

As you work through the Python assignments you will be building a set of scripts that will generate and validate the headers in your source code files. While you should find this useful to avoid losing points on your submissions, it's important to understand that this is only a vehicle to achieve a different set of objectives. In particular, the primary goals of this set of projects are to

- Expose you to most, if not all, of the Python concepts needed for the later ECS projects.
- Get you familiar with the Software Specifications that you will be expected to adhere to.
- Get comfortable with Jack-like object-oriented programming.
- Be comfortable with how the Jack OS launches programs.

Given these aims, some of the requirements may seem extraneous and/or unnecessary if viewed only from the viewpoint of building a header generation and validation tool. Please be patient and accept that there are reasons why the assignments have been structured as they are.

In this first assignment, we will be focusing on how to build a program using multiple classes and how to write purely object-oriented programs. Keep in mind that Python is not Jack and Jack is not Python – therefore there will be some things that we simply can't do the same way in both languages.

As you will see in the last few ECS projects, when a Jack program is written it is a collection of classes and no code or data exists outside of a class. Furthermore, each class is defined in a separate source code file in which the name of the class is the same as the name of the file. We will adopt that constraint for our Python programs, with the exception of our bootstrap code.

In the ECS project, once the separate Jack source code files are compiled they exist as separate VM, or Virtual Machine source code files – i.e., each Jack file is compiled into a corresponding VM file with no interaction between different classes. These are then translated into a single Assembly Language file by the VM Translator. In doing so, the VM Translator inserts some "bootstrap" code whose purpose is to initialize the Operating System libraries and launch the User's program. To do this last task, the VM Translator requires that the Jack code contain a class called "Main" and that it contain a function called "main". Many languages, most notably C, use a very similar approach. But Python doesn't – as a scripting language, there is no compiler

involved and, hence, no need to have such an "agreement" in place; you run a script and it starts executing that script from top to bottom.

The approach we will take will therefore be somewhat of a kluge. While your code will be required to have a "Main" class and a "main" function within that class, this would create the situation that the command line name for all of your programs would be "Main". While this has some very nice implications for grading, it is not consistent with the software requirement of the ECS projects, which have specific command line names for the various programs you will write. We will kill two birds with one stone by having a top-level script that is named using the desired command line name and which serves as our "bootstrap code"; this script will not contain a class definition but rather a very, very short script that does little, if anything, more than invoking the Main class' main function.

If you are familiar at all with Python (or even most other languages) you will notice that the requirement that each program have a Main class that is contained in a file called Main.py means that it is not possible to have multiple programs in the same directory, which would allow them to easily share classes that they both utilize. While unfortunate, this is also true of the Jack programs that we are trying to mimic. If you want to use classes in multiple projects, either Python or Jack, then you will need to place copies of them in each program's directory. Furthermore, all of your Python files – as distinguished from the built-in modules – that make up your programs must be in that program's directory; again, this is simply a mirroring of the constraints you will have when you switch over to Jack.

Notice that the class Main is capitalized. Windows users are generally not used to program names being case sensitive. But remember that not only is Python case sensitive, but that the file systems on *nix platforms are as well and your code is expected to run on both sides of that divide. Thus, pay attention to case when saving your files. If you need to change the case of an existing file on a Windows platform, you may run into problems depending on exactly how you try to do it since the OS may see that you are trying to rename the file to itself and do nothing. If this is the case, you can generally use a two-step process in which you first change the name to the desired case but also change it in some other way, such as putting an underscore in front of the name. Then you rename it again removing the underscore.

Arguably, it is best to adopt a "lowercase-only" approach, but the simple fact is that the ECS authors did not do so and we have to deal with it. By doing so in the Python assignments, you can figure out how to handle it before you get to the ECS projects.

Using our strategy, the ubiquitous Hello World! program takes the following form:

File: Hello.py

```
from Main import Main

Main.main()
```

File: Main.py

```
class Main:

    def main():
        print("Hello World!")
```

For your first program, you are to write a program named "py01" that instantiates a Header object and prints out the proper header for each of the file types covered in the Header Format Guidelines. The idea is that when you write a Python program to generate one of these files, you can simply have this object return a suitable header that can then be printed to the file immediately after creating it. If a file has defined header, the object should return an empty string.

The content of Main.py is shown on the next page. Aside from updating the file with your UserID and Programmer information, this file should be used as is.

Your task is to write the Header class so that it is compatible with Main. How you do this is up to you. In future projects you will be given interface specifications for all public functions in the classes you develop. For this project, the specifications will be included provided as part of the solutions.

The expected output of this script follows the listing of Main.

Contents of file Main.py

```
from Header import Header

class Main:

    def main():

        header = Header()

        header.setDividerLength(60)
        header.setValueColumn(20)
        header.setUserID("ADET_ID")
        header.setProgrammer("Lname, Fname MI.")
        header.setCourse("CSCI-400")
        header.setTerm("SP14")
        header.setProject("01")
        header.setPython("3.3.3")

        print("*****************************************")
        print("***       Header Format Examples       ***")
        print("*****************************************")
        print()

        for ext in ["hack","xml","hdl","asm","vm","jack",”py”]:
            print("---------------")
            print(ext.upper())
            print("---------------")
            print(header.gen("AnExample." + ext), end='')

        print("*****************************************")
        print("***   End of Header Format Examples    ***")
        print("*****************************************")
```

Output from py01.py

```
*****************************************
***       Header Format Examples       ***
*****************************************

---------------
HACK
---------------
---------------
XML
---------------
---------------
HDL
---------------
//=============================================================
// USERID:........ ADET_ID
// PROGRAMMER:.... Lname, Fname MI.
// COURSE:........ CSCI-400
// TERM:.......... SP14
// PROJECT:....... 01
// FILENAME:...... AnExample.hdl
//=============================================================

---------------
ASM
---------------
//=============================================================
// USERID:........ ADET_ID
// PROGRAMMER:.... Lname, Fname MI.
// COURSE:........ CSCI-400
// TERM:.......... SP14
// PROJECT:....... 01
// FILENAME:...... AnExample.asm
//=============================================================
```

```
--------------
VM
--------------
//============================================================
// USERID:........ ADET_ID
// PROGRAMMER:.... Lname, Fname MI.
// COURSE:........ CSCI-400
// TERM:.......... SP14
// PROJECT:....... 01
// FILENAME:...... AnExample.vm
//============================================================

--------------
JACK
--------------
//============================================================
// USERID:........ ADET_ID
// PROGRAMMER:.... Lname, Fname MI.
// COURSE:........ CSCI-400
// TERM:.......... SP14
// PROJECT:....... 01
// FILENAME:...... AnExample.jack
//============================================================

--------------
PY
--------------
#============================================================
# USERID:......... ADET_ID
# PROGRAMMER:..... Lname, Fname MI.
# COURSE:......... CSCI-400
# TERM:........... SP14
# PROJECT:........ 01
# FILENAME:....... AnExample.py
# PYTHON VERSION:. 3.3.3
#============================================================

****************************************
***   End of Header Format Examples    ***
****************************************
```