Design doc for hw3-boss

Problem (what): Employee data files (employee\_data1 and employee\_data2) contain all necessary data but are not correctly formatted for new HR database. (why): the new database offers no value without properly formatted data

Solution: Python script that operates on an input file with the old data format and produces a new file with name (oldfilename)\_new and contains all the same data but with all needed formatting fixes applied.

Format fixes needed: 1) Single name column needs to become two columns, left-most with first-name, right-most with last-name. (A quick VBA macro check was used on the sample data file to verify that none of the old “name” data has more than two words (i.e. no middle names, suffixes, etc.), so te script will simply use the last word in the cell as the last name, and consider any other content the first name.) 2) Ensure date of birth is in mm/dd//yyyy format (with zeros). The sample data set has mm/dd/yyyy format with no leading zeros, and some previous data may be in yyyy-mm-dd format (with zeros). The script will handle both possibilities. 3) The Social Security numbers will be converted from fully shown to masked-last-4 format (i.e, the format will be \*\*\*-\*\*-####), \* and – characters will be retained. 4) The state column will be converted from fully written out values to two-letter standard abbreviations (e.g. “Idaho” will become “ID”. Because the current data set is in use, it will be assumed that all entries for date, SSN, and state are valid (no special content checks will be included.) However, we will put in a simple validity check for lines so that, if something unexpected happens (like the user types the wrong file-name), conversion operations are skipped and the user is notified that there is bad data. Just like the old data set, the data in the new data set will be in continguous rows and columns, with column headers at row 1 matching exactly the previous column headers. The program will blindly copy the header row, it will not check that this row makes any sense.

Design considerations: In order to deal with large file sizes without the need for a lot of memory usage, the script will open read and write files simultaneously, and perform a line-by-line conversion. The script will prompt for the filename of the file to convert, and abort conversion if a destination file with the same name already exists. Because only a small number of file conversions are envisioned, the user will be prompted for filenames one at a time, with a simple blank response being the cue to terminate execution.

Procedural outline: The program will P) prompt user for filename, check for existence of file, and check for no existing file with the (filename)\_new name. If an error condition is detected, the user will be prompted for a new filename. If the user simply presses return, the prompt will indicate that this is the signal to terminate execution. On receiving a valid filename, the program will O) open the file for input, and create a new file with (filename)\_new for output. It will then C) (convert) read one line of input, parse the line to identify the data fields, generate a set of new data fields for writing, combine these into a single line, and then write these fields to the new file. It will then automatically close the files and return to the prompt (P).

Detailed procedures:

P: Prompt

P1: Display opening message to alert user that it is running, and explain how to terminate via simply pressing enter

P2: Initialize valid-entry to false, and keep-asking to true

P3: While keep-asking: (do rest of P, O, and C)

P4: filename = input - prompt message

P5: If filename = None, set keep-asking to false

P6: Valid-file-read = true if …

P6.1: try: open-filename for read, close filename; except FileError: print – file not found.

P7: If Valid-file-read

P7.1 Initialize: dest-filename = filename & “\_new”; valid-file-write = false

P7.2 try: open-dest-filename for read, print – destination file already exists, close dest-filename; except FileError: try:

P7.2.1 open dest-filename for write; valid-file-write = true, close dest-flename

P7.2.2 except FileError: print – unable to open dest-filename for writing

P8: Valid-entry = Valid-file-read AND Valid-file-write

O: File handling

O1: If (valid-entry): (do O2 and C)

O2: with open filename for read as Read\_File, dest-filename for write as Write\_File (do C) – use csv conventions with comma-delimiter

C: Converting

C1: Copy headers

C1.1 Header\_Line = read one line in filename

C1.2 write Header\_Line to dest-filename

C2: For Data\_Line in filename:

C2.1: Read one line in filename

C2.2 If Len(Data\_Line) == 4 (do rest of C2, else print – warning: invalid line found, check file

C2.3 Convert Old\_Name to New\_Name (function) (1 in, 2 out)

C2.4 Convert Old\_DOB to New\_DOB function

C2.5 Convert Old\_SSN to New\_SSN function

C2.6 Convert Old\_State to New\_State function

C2.7 New\_Data\_Line = Compose the new line by joining with commas

C2.8 Write New\_Data\_Line to dest-filename

Functions

C2.2: Parse: Python csv automatically does this for us, one index per list, so no code needed

C2.3: Convert Name: In: Data\_Line[0] Out: New\_Name [0:1]

C2.3.1: New\_Name = split(Data\_Line[1:3],’ ‘)

C2.4: Convert DOB: In: Data\_Line[1], internal name dob Out: New\_DOB

C2.4.1: If Find(dob, ‘/’: -- decide if we have m/d/yyyy or yyyy-mm-dd

C2.4.1.1: (if true): dob\_split = split(dob,’/’)

C2.4.1.2: (else): (parse and re-arrange elements of DOB in the dashed format)

C2.4.1.2.1: dob\_split\_temp = split(dob,’-‘)

C2.4.1.2.2: dob\_split[0] = dob\_split\_temp[1]

C2.4.1.2.3: dob\_split[1] = dob\_split\_temp[2]

C2.4.1.2.4: dob\_split[2] = dob\_split\_temp[0]

C2.4.2: try:

C2.4.2.1: if int(dob\_split[0]) < 10 AND dob\_split[0] != 0

C2.4.2.1.1: (if true) – dob\_split[0].insert “0” at start

C2.4.3: except ValueError: dob\_split[0] = “00”

C2.4.4: try:

C2.4.4.1: if int(dob\_split[1]) < 10 AND dob\_split[1] != 0

C2.4.4.1.1: (if true) – dob\_split[1].insert “0” at start

C2.4.5: except ValueError: dob\_split[0] = “00”

C2.4.6: return join(dob\_split,’/’)

C2.5: Convert SSN In: Data\_Line[2], internal name: ssn, Out: New\_SSN

C2.5.1: return “\*\*\*-\*\*-“ & ssn[8:12]

C2.6: Convert State In: Data\_Line[3], internal name, state\_name, Out: State\_Abbr, uses a static state data key (STATE\_DATA\_KEY)

C2.6.1: for long\_name: abbr in STATE\_DATA\_KEY:

C2.6.1.1 if long\_name == state\_name return abbr