Process Employee Information Script Documentation

Existing Code and Future Implementation Specifications

# Objective

The main purpose of the script is to take raw employee information, as a JSON file, and generate a formatted JSON file for Database (DB) Admins so they can easily enter new employees into the DB. There is some existing code available from a previous team that was hired on contract, but the project has been moved to work internally. The goal is to fix the existing code and add the specifications detailed below and in the user stories to complete the script. Following good programming practice, every function needs to have its docstring and needs to include comments to explain the code within it. There should be no code outside of any function, and no print statements unless requested as a functionality.

A solution that makes use of RegEx is preferred and will gain additional marks.

# Existing Code

The following methods have already been written. Please ensure they follow the specifications detailed below and are working as intended before adding new functionalities.

main.py

main()

* Arguments: none.
* Return: none.
* Functionality: invokes the code necessary for the script to run as intended. As of right now, it only calls check\_path.

usr\_input.py

check\_path(file\_path, data\_files)

* Arguments:
  + file\_path (str) – The file path to the JSON, or folder.
  + data\_files (list) – A list to store the valid file paths, initially set to empty list.
* Return:
  + int – An integer of 1 if the path given does not exist, or an int of 2 if the file path is a formatted JSON file. These int values act as error codes.
  + tuple – A tuple of the valid file paths that can be parsed.
* Functionality:
  + Need to check if the given file path exists, and if not display the message "<file path> does not exist." and return an int of 1.
  + If the given file path exists and is a folder, check\_path need to check the content of the folder for any valid files. This should work even if there are nested folders.
  + If the given file path exists and is a file but is already formatted JSON file, it needs to be skipped and an int of 2 returned. The names of already formatted JSON files end with '\_formatted.json'.
  + If file(s) within the folder are valid files to be processed, they need to be returned in the tuple, so that each element in the tuple consists of the complete file path followed by the file name.

# New Code

The following functions were planned to be created by the previous team to meet the specifications of the script.

main.py

error\_handle(check\_return)

* Arguments:
  + check\_return (int or tuple) – The return value of the check\_path function.
* Return: None.
* Functionality:
  + If the return value is 1, the script should exit because the path given was not valid. No message must be displayed here, as the message was already displayed in check\_path.
  + If the return value is 2, the script should display the message "The file provided is already processed.” and exit because the file given should not be processed.
  + If the tuple returned contains no paths, the script should display the message "There are no valid files to process in the folder provided." and exit because there are no valid files to process.

Note: Find out information about sys.exit() and use it to exit the script.

print\_output(num\_files, num\_emps)

* Arguments:
  + num\_files (int) – The number of valid files that were processed.
  + num\_emps (int) – The number of valid employee entries that were processed.
* Return: None.
* Functionality: Display the number of files and employees processed in the format seen below (the actual numbers are for illustration only – they will change depending on the situation).

|  |
| --- |
| ============================================================  ---------------------Processing Summary---------------------  ============================================================  Number of files processed: 1  Number of employee entries  formatted and calculated: 2 |

start\_process(tup)

* Arguments:
  + tup – Tuple of valid JSON file paths generated from check\_path.
* Return: None.
* Functionality:
  + Calculate the number of files and employees processed.
  + Process each file and save the information into a new formatted JSON file as detailed in parse\_file.py.
  + Print the output information as detailed in print\_output.

parse\_file.py

get\_json\_content(file)

* Arguments:
  + file (str) – The path to the JSON file to get the content of.
* Return:
  + list – The content of the file passed into the function. Should be a list of dictionaries, where each dictionary is the employee entry.
* Functionality: Read the JSON file that was passed in, to get the data in the file.

generate\_email(first\_name, last\_name)

* Arguments:
  + first\_name (str) – The employees first name.
  + last\_name (str) – The employees last.
* Return:
  + str – The company email in the format of <first letter of the first name><full last name>@comp.com all in lower case. For example, John Smith’s email would be jsmith@comp.com.
* Functionality: Generate the email address for the employee entry to follow the format detailed above.

generate\_formatted\_file(emp\_list, orig\_path)

* Arguments:
  + emp\_list (list) – A list of dictionaries where each dictionary is the employee entry after being formatted as detailed in process\_each\_emp.
  + orig\_path (str) – The file path of the original file that was passed in.
* Return: None.
* Functionality: For each file save its content to a separate new JSON file in the same folder where the original file is located, and that has the original file name with "\_formatted.json" appended at the end.

generate\_salary(job\_id, state)

* Arguments:
  + job\_id (str) – The job ID of the employee. The job ID could be SA\_REP, SA\_MNG, HR\_REP, HR\_MNG, IT\_REP, or IT\_MNG
  + state (str) – The US state the employee is located in.
* Return:
  + int – The calculated salary of the employee.
* Functionality: Generate the salary of the employee based on the following criteria:
  + Depending on the department, the employee has a given base salary:
    - SA (Sales) - $60,000
    - HR - $70,000
    - IT - $80,000
  + If the employee is a manager (MNG), the employee gets an extra 5% on top of the base salary.
  + If the employee lives in the following states: New York, California, Oregon, Washington, Vermont, they get an extra 1.5% on top of the base salary and on top of their manager’s base salary if they're also a manager. Use the following values for the above mentioned 5 states: NY, CA, OR, WA, VT.

Note: the salary calculation will produce a float value, instead of an intended int value. Ensure the returned salary is a correct int corresponding to the obtained float value. For example, if the salary is 60199.999999 then the returned salary should be 60200; if the salary is 74550.0 then the returned salary should be 74550.

process\_each\_emp(emp\_list)

* Arguments:
  + emp\_list (list) – The list of employee entries extracted from the JSON file.
* Return:
  + list – A list of dictionaries where each dictionary is each employee entry.
* Functionality: Process each employee entry and format it based on the following criteria.
  + Ensure the phone numbers and zip codes are valid US phone numbers and zip codes. If any of them is invalid, skip the entry. If they are both valid, save them to the dictionary as an int.
  + Remove the last entry of the dictionary as that is extra data that’s not needed.
  + Ensure the first name, last name, address line 1 & 2, city, and job title have proper casing where the first letter of each word is capitalized, and all other letters are lower case. Remove any extra spaces in all of these fields. If within Address Line 1 or Address Line 2 there is a word starting with a digit, such word should be lowercased.
  + Generate the company email and add it to the dictionary with the key “Company Email”.
  + Generate the salary and add it to the dictionary with the key “Salary”.

validate\_phone\_number(phone\_number)

* Arguments:
  + phone\_number (str) – The phone number of the employee.
* Return:
  + int – If the phone number is invalid, the number 1 to indicate an error. If the phone number is valid, the 10-digit number as an int.
* Functionality – Check the given phone number to ensure it is a 10-digit number with no extra characters, removing any leading and/or trailing spaces. If the phone number is invalid, display the message "<phone\_number> is not a valid US phone number, skipping this employee entry..." and return 1.

validate\_zip(zip\_code)

* Arguments:
  + zip\_code (str) – The zip code of the employee.
* Return:
  + int – If the zip code is invalid, the number 1 to indicate an error. If the zip code is valid, the 5-digit number as an int.
* Functionality – Check the given zip code to ensure it is a 5-digit number with no extra characters, removing any leading and/or trailing spaces. If the zip code is invalid, display the message   
  "< zip\_code> is not a valid US zip code, skipping this employee entry..." and return 1.

usr\_input.py

get\_usr\_input(msg)

* Arguments:
  + msg (str) – The message to prompt the user.
* Return:
  + str – The user’s input.
* Functionality – Prompt the user with a given message and get their input. The message should be the original message passed in with a colon and space at the end as seen below.

|  |
| --- |
| Please enter the path of the file or the folder containing the files: |