Process Employee Information Script Documentation

Existing Code and Future Implementation Specifications

# Objective

The main purpose of the script is to take raw employee information, either as a JSON or PDF 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.

# 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 checkPath.

usrInput.py

checkPath(filePath, dataFiles)

* Arguments:
  + filePath (str) – The file path to the JSON, PDF, or folder.
  + dataFiles (list) – A list to store the valid file paths.
* Return:
  + int – An integer of 1 if the path given does not exist. This is to act as an error code.
  + tuple – A tuple of the valid file paths that can be parsed.
* Functionality:
  + Need to check if the file path given even exists and return an int of 1 if not.
  + If the file does exist, need to ensure the file is not an already formatted JSON file. Already formatted JSON files need to be skipped.
  + If the file does exist and is a PDF or a JSON file, then they are valid files to be processed and need to be returned in the tuple.
  + If the file does exist and is a folder, need to check the contents of the folder for any valid files. This should work even if there are nested folders.

# New Code

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

main.py

errorHandle(checkReturn)

* Arguments:
  + checkReturn (int or tuple) – The return value of the checkPath function.
* Return: None.
* Functionality:
  + If the return value is 1, the script should exit because the path given was not valid.
  + If the tuple returned contains no paths, the script should exit because there are no valid files to process.

printOutput(numFiles, numEmps)

* Arguments:
  + numFiles (int) – The number of valid files that were processed.
  + numEps (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.

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

startProcess(tup)

* Arguments:
  + tup – Tuple of valid JSON or PDF file paths generated from checkPath.
* 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 parseFile.py.
  + Print the output information as detailed in printOutput.

parseFile.py

getJSONContent(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.

generateEmail(firstname, lastname)

* Arguments:
  + firstname (str) – The employees first name.
  + lastname (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.

generateFormattedFile(empList, ogPath)

* Arguments:
  + empList (list) – A list of dictionaries where each dictionary is the employee entry after being formatted as detailed in processEachEmp.
  + ogPath (str) – The file path of the original file that was passed in.
* Return: None.
* Functionality: Save the content of the list to a new JSON file to the folder the original file was located and that has the original file name with “\_formatted.json” appended at the end.

generateSalary(jobId, state)

* Arguments:
  + jobId (str) – The job ID of the employee. The job ID could be IT\_REP, IT\_MNG, HR\_REP, HR\_MNG, SA\_REP, or SA\_MNG
  + state (str) – The US state the employee is located in.
* Return:
  + float – 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:
    - Sales - $60,000
    - IT - $80,000
    - HR - $70,000
  + If the employee is a manager, the employee gets an extra 5% on top of the base salary.
  + If the employee lives in the following states, they get an extra 1.5% on top of the base salary and their manager salary if they’re also a manager.
    - New York, California, Oregon, Washington, Vermont.

processEachEmp(empList)

* Arguments:
  + empList (list) – The list of employee entries extracted from the JSON or PDF 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 they are invalid, skip the entry. If they are 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. Also ensure there are no extra spaces.
  + Generate the company email and add to the dictionary.
  + Generate the salary and add to the dictionary.

validatePhoneNumber(phoneNumber)

* Arguments:
  + phoneNumber (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.

validateZips(zipCode)

* Arguments:
  + zipCode (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.

usrInput.py

getUsrInput(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 a value: |

DocAnalysis.py NOT NEEDED

This must be done after writing the other 3 scripts and then integrated in them to fulfil the user stories.

establishConnectionAzure()

* Arguments: None.
* Return:
  + The client information after establishing the connection to document analysis.
* Functionality: Connect to document analysis using the correct endpoint and key.

getKVPairs(result)

* Arguments:
  + result – The result from running the analysis on the PDF file.
* Return:
  + list – A list of dictionaries with each dictionary being an employee’s information.
* Functionality: Extract the key value pairs from the analysis result set.

getPDFContent(document\_analysis\_client, file)

* Arguments:
  + document\_analysis\_client – The connection information to Azure.
  + file (str) – The file path to the PDF file to be analyzed.
* Return:
  + The result full result set from analyzing the PDF file.
* Functionality: Read the PDF file and run the correct analysis model to get key value pairs out of the PDF file.