Extraction of Attributes from Unstructured Documents (CV/Resumes)

**INTRODUCTION**

**Modules**: Simply, a **module** is a file consisting of **Python** code. It has an extension of (.py).

Our system is built by combining several user- defined modules as well as few in-built modules.

Executing the program is effectively executing the module, which is done by the following command in the terminal:

$**python name\_of\_module.py**

* List of Modules Present in the System
  + 1. **Main\_Console.py**

This module is responsible for taking all the .txt files from the given directory and then calling the Extract\_Resume function present in Extract\_Resume module for each of the .txt files.

* + 1. **Extract\_Resume.py**

This module contains the Extract\_Resume Function, which helps in extracting various attributes from the unstructured documents (Resumes/CVs) and writing them on an extracted\_output file (.txt).

* + 1. **Helper\_Functions.py**

This module contains all the functions which process the text file and extracts specific attributes. These functions are used by the Extract\_Resume function.

* + 1. **UPDATE.py**

This is an isolated module which gives the user the option to update the list of Skills (Technical & Soft), Roles & Responsibilities.

Along with these 4 modules, we have also used 6 user-defined ‘Dictionary’ in our system.

Though ‘Dictionaries’ are special data structures which are predefined in python, here by the term ‘Dictionary’ we mean modules containing lists of keywords which will help us to match specific attributes.

* List of Dictionaries Present in the System

**i) LANG\_DICT.py**

This Dictionary contains lists of spoken languages, programming languages, technical skills, roles and soft skills.

**ii) LOC\_DICT.py**

This Dictionary contains lists of all possible locations (in India).

**iii) INDUS\_DICT.py**

This Dictionary contains lists of all possible industries.

**iv) DEG\_DICT.py**

This Dictionary contains lists of all possible degrees.

**v) DESIG\_DICT.py**

This Dictionary contains lists of all possible designations in all industries.

Additionally, a module named **‘REGEX\_LIB.py’** has been used to store all regular expressions used for extracting the attributes.

**DESCRIPTION**

**1. Main\_Console.py**

Algorithm:

* Import the in-build ‘time’ Module [to use functions that could calculate time in various formats]
* Start\_time of the program is measured by calling time() function.
* Extract\_Resume module is imported so that its function ‘Extract\_Resume()’ can be used.
* Variable ‘mypath’ contains the path of the directory where the input resumes in text format are present.
* Variable ‘outputpath’ contains the path of the directory where the extracted outputs will be placed.
* ‘listdir’ class is imported from ‘os’ Module, which will list all the directories in a given path.
* ‘isfile’ class and ‘join’ class is imported from ‘os.path’ Module, which can test whether the given path is of a valid file or not.
* ‘files’ list is created which will contain the list of all the valid files present in the directory.
* For each file present in the ‘files’ list, call the Extract\_Resume function by passing the file name as argument.
* Print the difference between start\_time and current\_time to get the total execution time of the program.

Functions:

1. time.**time**()

**Arguments**: NULL

**Return Value:** Returns the time in seconds since the epoch as a floating point number.

2. listdir(path)

**Arguments: path** -- This is the directory, which needs to be explored.

**Return Value**: returns a list containing the names of the entries in the directory given by path.

3. isfile(path)

**Arguments: path** -- This is the address of directory/file, which needs to be checked.

**Return Value**: Return True if path is an existing regular file.

4. join(path, \*paths)

**Arguments:** Join one or more path components intelligently.

**Return Value**: the concatenation of path and any members of \*paths with exactly one directory separator (os.sep) following each non-empty part except the last, meaning that the result will only end in a separator if the last part is empty. If a component is an absolute path, all previous components are thrown away and joining continues from the absolute path component.

5. Extract\_Resume(filepath,outputpath)

**Arguments:** filepath—address of the input resume file.

outputpath-- address of the output extracted file.

**Return Value:** NULL

**2. Extract\_Resume.py**

Algorithm:

* Import ‘Helper\_Function’ Module for using functions which will extract specific attributes from the unstructured document.
* Open the provided file in ‘read’ mode.
* Split the file\_path into separate words and store the last word in the variable ‘name’ [It will help us to create unique output extracted files].
* Define a List ‘L’ in which all the lines present in the input file will be copied. [Instead of passing the input file to separate functions, passing the list ‘L’ will be easy for processing].
* Append empty lines at the start and end of the List ‘L’ to avoid errors like ‘list\_index\_out\_of\_range’.
* Call the function find\_EDG() from Helper\_Functions to extract Email, DOB and Gender from the unstructured document and store them in 3 separate variables.
* Call the function find\_np\_lo\_s() from Helper\_Functions to extract Notice Period, Location, Salary and Total Experience from the unstructured document and store them in 4 separate variables.
* Call the function find\_L\_I\_P() from Helper\_Functions to extract Spoken Languages, Industry and Position (Designation) from the unstructured document and store them in 3 separate lists.
* Call the function find\_qual() from Helper\_Functions to extract the academic qualifications from the unstructured document and store them in a single list.
* Call the function find\_exp() from Helper\_Functions to extract total work periods from the unstructured document and store them in a single list.
* Call the function find\_skill() from Helper\_Functions to extract Programming Languages, Technical Skills, Soft Skills and Roles-Responsibilities from the unstructured document and store them in 4 separate lists.
* Pass all the stored lists into the function write\_output() (defined in Helper\_Functions), to write the extracted attributes in a output file.

Functions:

1. open(name[, mode[, buffering]])

**Argument:** most commonly used with two arguments open(filename, mode).

**Return Value:** returns a file object

2. split(s[, sep[, maxsplit]])

**Argument:** string s and optional parameters sep & maxsplit.

**Return Value**: Return a list of the words of the string s.

3. append()

**3. Helper\_Functions.py**

Algorithm:

* Import all the 5 dictionaries: LANG\_DICT, LOC\_DICT, INDUS\_DICT, DEG\_DICT, DESIG\_DICT and also the Regular Expressions library REGEX\_LIB.py
* **Find\_EDG() function**:

This function takes the list ‘L’ containing all the lines of the unstructured documents as input and returns the attributes email, dob and gender.

**a)** Define an empty list ‘E’ for storing the email ids, two empty strings ‘D’ and ‘gender’ for storing the DOB and gender attribute respectively.

**b)** For each and every line in the list ‘L’, search whether there is an email-id present in the line by using the pre-defined regular expressions and the search() function from the ‘re’ library. [*Refer to the section below for usage of re.search()*]

**c)** For each and every line in list ‘L’, check whether the keywords ‘Birth’ or similar variations of this keyword are present. If the keyword is present, then search for various formats of dates in that current line and return them in the ‘D’ string. If the current line contains no dates, then perform the same search in the next line of the list. Return the matched date to be stored in the ‘D’ string. For returning the matched string, use the group() function from the ‘re’ library. [*Refer to the section below for usage of group()*]

**d)** Similarly, for each line in list ‘L’, search for the gender using the pre-defined regular expression from the REGEX\_LIB and if match is found, return in the designated string ‘gender’.

**e)**  Return the List of emails ‘E’ and the strings ‘D’ [for DOB] & ‘gender’ [for gender].

* **find\_qual() function:**

This function takes the list ‘L’ containing all the lines of the unstructured documents as input and returns the qualification attribute.

**a)** Define an integer variable ‘indx’, which will store the line number of the header of academic section in the unstructured document and initialize it to 0.

**b)** Define an empty list ‘NL’ to store the lines of the file starting from academic details till the end [defined to be after 10 lines from the header].

**c)** Define another empty list ‘q’, which will be used to return the academic details from the function.

**d)** Using the regular expression which matches the header of academic section in the unstructured document, search for header of academic section and store its index in the variable ‘indx’.

**e)** If theproposed end point (header index + 10) exceeds the actual end point of the file, then set the end point to the end of the file.

f) Copy all the lines from the list ‘L’ to the list ‘NL’, staring from ‘indx’ till the end point.

g) For each line in list ‘NL’, check whether possible spurious keywords like ‘MS Office’ or ’RPG Cables’ are present or not. If these keywords are absent, then check for keywords which occur in pre-defined degree dictionary.

In case of Degree keyword match in line, append the entire line in the list q.

h) Return the list q

* **find\_np\_lo\_s () function**

This function takes the list ‘L’ containing all the lines of the unstructured documents as input and returns the attributes Notice Period, Location and Salary.

a) Define three empty strings ‘np’, ‘loc’ and ‘te’ for storing the attributes Notice Period, Location and Total Experience respectively.

b) Define Lists ‘L2’ for storing lines containing possible Notice Period, L11 for storing lines containing possible salaries and a list ‘sal’ for storing salaries.

* **Sgsf**
* **Sfgsf**
* **sgsf**