International Journal of Innovative Mathematics, Statistics & Energy Policies 5(4):1-9, Oct.-Dec., 2017

© SEAHI PUBLICATIONS, 2017 **www.seahipaj.org** **ISSN: 2467-852X**

A Surjective-Function Based Algorithm for Generating Senate Format

Richard O. Akinola, Stephen Y. Kutchin, Homti Nahum, Izang. A. Nyam, Kehinde J. Ajibade and Ayodeji S. Ayodele

Department of Mathematics, Faculty of Natural Sciences, University of Jos, P. M. B. 2084, Jos -Plateau State, Nigeria

Corresponding Author: R. O. Akinola (roakinola@gmail.com)

ABSTRACT

In this paper, we state with proof a Mathematical theorem and apply it in presenting an algorithm for generating senate format in the University of Jos. Though the function used is not injective due to the complexity of the semester results, however, it is surjective. The proposed algorithm is computationally efficient, uses only soft copies of session-al results of each level and solves the age-old problem of manually generating the senate format using hard copies of session-al results. No need for lecturers to move from one department or faculty to another trying to collect hard copies of results before generating the senate format. The algorithm is not only applicable to the University of Jos as it can be adapted for use in any other Nigerian university. The algorithm and an example is given which shows the workability of the program.

Keywords:

1.0 INTRODUCTION

At the end of each academic session in the University of Jos, Nigeria, one of the problems faced by level coordinators is that of preparing the senate format. The senate format is a document usually in openoffice or excel format prepared by each level coordinator and submitted to the University senate at the end of the year, showing the list of students in a particular level and the marks obtained in all courses undertaken by them. It also states the name, matriculation number, mode of entry, number of semesters spent, Cumulative Grade Point Average (CGPA) and remark of each student. The remark column states which courses are to be repeated if a student fails one or more courses, if a student is on probation or withdrawn. A student is on probation if his/her CGPA is less than 1.00 at the end of a particular session. However, such a student is withdrawn if after two successive sessions, his/her CGPA is below 1.00.

No matter the number of students in each level, the problem of collating each student's marks and GP in each course for a large number of students is herculean and painful to say the least. It involves each level coordinator going from one department to another to collect the marks and corresponding GP of each student in all courses and typing the result of each student in excel format. 'Woe' betide such a lecturer if the number of students are more than one hundred. Usually each student takes at least ten courses per session. The number of courses taken by a student for an entire session depends on the number of carry-overs, he/she passing the co-requisites and pre-requisites. Tabulating the result manually into an excel spreadsheet is laborious and most importantly subject to errors due to parallax and typos. Many a student have been at the receiving end of such 'dangerous' typos which are avoidable. In this paper, we present an efficient algorithm for compiling senate format for a particular level given that the result of all the courses taken by all the students are in the same folder. The results need not be in the same folder or directory as long as path to their where they are located is specified in the program. This algorithm will put a smile on lecturers/level-coordinators faces, because what should normally take them days or weeks can be done in a matter of minutes! Besides, no more need for hard copies of results and we hope to save our forests.

As far as we know not much has been done or written on this subject since each University has its own

peculiarities in terms of what the senate expects from the level-coordinators at the end of the session. In [1], the author presents an octave based algorithm for computing the Cumulative Gross Point Average of students at the University of Jos. In that paper, the algorithm depends on typing the GP of each student in all the courses taken and then outputs only the CGPA. However, the algorithm did not describe how to collate the marks and GP of the courses. Akinola in [2] presents a computationally efficient algorithm for computing the CGPA of a large number of students, with the algorithm outputting the remark column. Nevertheless, the algorithm though powerful and an improvement on [1], depends on typing individual result manually. In the present work, we solve the problem of manually collating the results and GP of each student no matter how many they are and irrespective of the number of courses taken for that particular session.

The plan of this paper is as follows: in Section 2.0, we describe the methodology for solving the problem by presenting Theorem 1 which is at the heart of the algorithm. This is then followed in Section 2.1 by describing a major tool used in the algorithm; which is python's dictionary. In Section 3.0, we present the algorithm and an example is given to illustrate its functionality. This is then followed by conclusion, references and an appendix is given.

2.0 METHODOLOGY

Our approach for solving this age old problem of preparing senate format at the University of Jos will be based on Python. Python is an object-oriented, high-level programming language [3, 4]. Other examples of high-level programming languages are C, C++, Java and Perl. Low-level programming languages are often-times termed machine or assembly languages. Thus, programs written in high level programming languages needs to be processed into low-level ones before they can run successfully. Hence, the extra time spent in processing is a 'small disadvantage' of high-level languages. However, high-level languages are portable, easier to code, shorter, more likely to be correct and easier to read. Programs written in Python are executed by an interpreter and that is why it is often regarded as an interpreted language [3]. The interpreter is used in two ways viz.: script and interactive mode. In the later mode, Python programs are typed by the user while the interpreter displays the result. In the script mode, you save the code in a script file before using the interpreter to execute or run the contents of the file. Python scripts are usually saved with names that end with .py. This means if you are working in an environment such as a UNIX command window and have a script named *filename.py*, you type **python filename.py**. For more information on Python and how to download it, visit http://python.org [3] or [5].

To illustrate the functionality of the program, we will use the result of seven courses viz-a-viz MTH201, MTH202, MTH203, MTH204, MTH205, MTH207 and MTH207; and seven students with abbreviated names as shown in Table 1. The total scores is from 0 to 100% while the grade points range from 0 to 5. Marks from 0 to 39 correspond to a G. P. of 0, total marks from 40 to 44 correspond to a G. P. of 1, from 45 to 49 corresponds to a G. P. of 2. In the same vein, marks from 50 to 59 correspond to a G. P. of 3, total marks from 60 to 69 corresponds to a G. P. of 4 while total marks from 70 to 100 corresponds to a G. P. of 5. This means that if a student obtained a total mark of 68, the corresponding G. P. will be 4 and so on and so forth. We state the following important result in use.

Theorem 1: Let X be the set of matriculation numbers of all n students that offered and sat for a particular course in a certain department and let Y be the set of pair of total score (%) in the particular course and grade points (0, 1, 2, 3, 4, 5); then the function f : X-->Y is not bijective but surjective.

Proof: In a particular course, every element in the domain X (the set of matriculation numbers) is unique because no two students have the same matriculation number. However, because it is possible for two or more students to have the same total score and G. P pair in the codomain Y, f is not injective but many to one. Since every element in the codomain Y will have a preimage in the domain and no element in X has no image in Y, f is surjective.

Next, we describe the file format containing the names of the students in a particular level, for the purpose of this paper, we will use seven students as shown in 1. However, there is no limit to the amount of students that should be in a particular level. The format for the file containing the student names, that is *names.txt* has three columns, S/N = Serial Number, Matriculation number and Full names of the students. We have used initials for ease of use.

S/N	MATRIC. NUMBER	NAMES
1	UJ/201X/NS/0018	A. M. A.
2	UJ/201X/NS/0096	R. G. Y.
3	UJ/201X/NS/0134	A. F.
4	UJ/201X/NS/0164	A. J. R.
5	UJ/201X/NS/0370	D. M. A.
6	UJ/201X/NS/0387	D. N.
7	UJ/201X/NS/0563	I. L. D.

Table 1: The file containing student names is in the following format. Care must be taken in writing the Matriculation number because it has to be correct, otherwise it will affect the compiled senate format.

2.1 FORMING THE DICTIONARY

In this subsection, we will make use of the result of Theorem 1 in forming the dictionary. This is because the python dictionary at the heart of the program takes in unique matriculation number as keys (X in theorem 1) and returns subsets of Y as defined in Theorem 1 as output. The function f in this context is any of the courses. Using the result of Table 2, for example, we form a python dictionary called MTH204 with matriculation number as keys and the pair (Total (100%), G. P) as values. However, because we want to tabulate the final output for a particular course, we concatenate them so that each enters a separate cell. This means that whenever you call MTH204[UJ/201X/NS/0001], it will return (17, 0); MTH204[UJ/201X/NS/0018] = (41, 1) and MTH204[UJ/201X/NS/0563] = (50, 3) all pairs are concatenated without the braces and comma.

Table 2: Table showing the abridged result of 563 students that took MTH204

S/N	MATRIC NUMBER	NAMES	CA 30%	EXAM 70%	TOTAL 100%	L. G	G. P	REMARK
1	UJ/201X/NS/0001	A. C. B	7	10	17	F	0	FAIL
2	UJ/201X/NS/0002	R. A. O	20	60	80	A	5	PASS
3	UJ/201X/NS/0003	A. J.	19	51	70	A	5	PASS
4	UJ/201X/NS/0005	S. K. Y	20	52	72	A	5	PASS
…	**…**	**…**	**…**	**…**	**…**	**…**	**…**	**…**
18	UJ/201X/NS/0018	A. M. A.	17	24	41	E	1	PASS
19	UJ/201X/NS/0019	N. H.	18	50	68	B	4	PASS
…	**…**	**…**	**…**	**…**	**…**	**…**	**…**	**…**
563	UJ/201X/NS/0563	I. L. D	7	13	20	F	0	FAIL

CA stands for Continuous Assessment; L. G. means Letter Grade, and G. P. means Grade Point. N. B. In general, all results take the above format.

Note that because the results are as tabulated in Table 2 are located in a particular directory, we need to first open the file. After opening the file as shown in the appendix, each file is then fed into the '*returndict*' function. Consider the results of MTH204 saved as MTH204.txt (after copying and pasting from a dot xls file). First, we open the file as follows: ***file = open('MTH204.txt', 'r').*** Because the *returndict* function is a 'function' as defined in python, it must return an output when called, in this case our output should be a dictionary. In this example, we assign MTH204 as the output viz-a-viz ***MTH204 = returndict(file).*** Wow!, we have formed the MTH204 dictionary. Hence, as mentioned earlier whenever you call MTH204[UJ/201X/NS/0001], it will return (17, 0); MTH204[UJ/201X/NS/0018] = (41, 1) and MTH204[UJ/201X/NS/0563] = (50, 3) all pairs are concatenated without the braces and comma.

#The function returndict takes as input a file in .txt or .csv format and returns a dictionary as output.

```
def returndict(file) :

    course = { }                    #course is pre-defined as an empty dictionary in python

    for line in file :

    line = line.rstrip()

    line = line.split('\t')

    if len(line) <  8 : continue   #This overlooks lines without student

                                   #details like matnumber etc

    #dictionary with matnumber as  key; and mark(%) and GP as key

    course[line[1]] = line[5] +"\t"+ line[7]

    return course
```

3.0 THE ALGORITHM AND COMPUTATIONAL EXAMPLE

Assuming all the dictionaries for MTH201, MTH202, MTH203, MTH204, MTH205, MTH207 and MTH209 have been formed, and the file names.txt containing the list of all students and their matriculation number (arranged in order of matriculation number) have been opened. We remark that in practice, the number of courses offered by students in a particular level is not less than 10, meaning, we form the dictionaries for all such courses. The program can effectively handle any amount of student. This is how the program works. It takes the already arranged matriculation numbers one by one. For the first matric number in Table 1, UJ/201X/NS/0018; it checks if this matriculation number is in the MTH201 dictionary, if yes, then it concatenates the total score and G.P. In the same vein, it checks if this matric number is in the MTH202 dictionary? If true, the total mark and G.P., are concatenated with the already recorded ones for MTH201, otherwise, the program concatenates two empty cells.

Again, the program checks if this matriculation number is in the MTH203 dictionary, if yes; it picks the pair (total mark, G. P) and concatenates them to the MTH201, MTH202 already concatenated, otherwise, inserts two empty tabs. This continues until all the open files containing different courses have been exhausted. Even if they are thirty courses, the program will check if this matric number is their respective dictionaries. After the first student's course records have been concatenated, the program picks the second matriculation number in this case, UJ/201X/NS/0096. The same process is repeated until all the matriculation numbers in the file names.txt have been exhausted. Hence, we have the generated senate format without errors saved in dot txt format. The result of the generated senate format for the seven students under consideration is as presented in Table 3.

Table 3: Generated senate format for a group of seven students

S/N	MATRIC. NUMBER	Names	MTH 201	GP	MTH202	GP	MTH203	GP	MTH 204	G P	MTH 205	G P	MTH207	GP	MTH209	GP
1	UJ/201X/NS/0018	A.M.A	18	0	42	1	46	2	41	1	40	1			31	0
2	UJ/201X/NS/0096	R.G.Y.	50	3	44	1	42	1	62	4	45	2				
3	UJ/201X/NS/0134	A. F.	15	0	24	0	41	1	23	0	27	0			30	0
4	UJ/201X/NS/0164	A. J.R.	60	4	64	4	58	3	77	5	86	5			54	3
5	UJ/201X/NS/0370	D.MA	40	1	31	0	40	1	21	0	40	1			29	0
6	UJ/201X/NS/0387	D. N.	55	3	52	3	64	4	52	3	63	4	45	2	61	4
7	UJ/201X/NS/0563	I. L.D.	7	0	21	0	19	0	20	0	15	0			29	0

Table 3: The table above shows the generated senate format for a group of seven students. We have limited the program to just seven students but in actual fact, most classes are larger than 30 students. Some classes are as large as 150 students in real life. Next, we pesent the algorithm.

Make sure that python is installed in your system and be sure you know the directory or folder in which all the results are kept.

THE ALGORITHM

1. Copy all the results from either .xlsx or .xls or .csv formats; paste and save them into .txt formats. For example, we copied the results of the following courses MTH201, MTH202, MTH203, MTH204, MTH205, MTH207 and MTH209 from their excel or OpenOffice files to MTH201.txt, MTH202.txt, MTH203.txt, MTH204.txt, MTH205.txt, MTH207.txt and MTH209.txt respectively.
2. Open each of the files as shown in the first 14 lines of the appendix.
3. Copy and paste the file containing matriculation number and names into names.txt.
4. Open the file 'names.txt' as shown in line 15 in the appendix.
5. Open the file '*200_senate.txt' for writing.*
6. *Save the file as senate_200.py.*
7. *Open a python shell*
8. *In lower case type 'python senate_200.py'.*
9. *If there is no error(s), the result will be saved in the file named '200_senate.txt'.*
10. *Open the file '200_senate.txt'. Copy and paste it to excel or OpenOffice. Save with appropriate name. Ooops, the end.*

Note that you can specify the path to the folder in which the output can be saved. Otherwise, the current folder or working directory houses the output.

CONCLUSION

We have presented a mathematical based algorithm for generating the senate format for any level at the University of Jos. This is a local solution to a local problem but can be adapted in other institutions where students progress can be monitored concurrently throughout their course of study. The algorithm is fast and will be applied by level coordinators to generate the senate format automatically without any manual effort. One of the strengths of the algorithm is that it allows the level coordinators to supply few missing results (when found) manually in cases where results are missing before the final computation for the Cumulative Grade Point Average and yearly summary. One of the strengths of the algorithm is that all results do not have to be in the same folder or directory as long as the path is correctly specified.

ACKNOWLEDGEMENT

The first author acknowledges the support of Professor L. S. O. Liverpool of the University of Jos who inspired me in writing the first paper [1] which gave birth to the second [2]. That inspiration brought about this current article. The first paper was written in octave while the last two was in python, for which I duly acknowledge the National Research Foundation of South Africa for providing funds for my post-doctoral research at the University of Cape Town where python programming skills were horned in solving problems.

REFERENCES

[1]. R. O. Akinola, An Octave-based algorithm for computing the Cumulative Grade Point Average of students.

[2]. R. O. Akinola, Computationally Efficient Algorithm for computing Cumulative Grade Point Average of a Large Number of Students, Afr J Comp & ICT - ISSN 2006-1781. Vol 7. No. 2 - June, 2014.

[3]. Python: http://python.org.

[4]. Downey A, Think Python: How to Think Like a Computer Scientist Green Tea Press, Needham, Massachusetts, http://www.thinkpython.com, 2012.

[5]. https://www.enthought.com/canopy-express/

APPENDIX

```
file = open('MTH201.txt', 'r')

MTH201 = returndict(file)


file = open('MTH202.txt', 'r')

MTH202 = returndict(file)


file = open('MTH203.txt', 'r')

MTH203 = returndict(file)


file = open('MTH204.txt', 'r')

MTH204 = returndict(file)


file = open('MTH205.txt', 'r')

MTH205 = returndict(file)


file = open('MTH207.txt', 'r')

MTH207 = returndict(file)


file = open('MTH209.txt', 'r')

MTH209 = returndict(file)


file = open("names.txt", 'r')          #file containing the names and matnumbers of students

out = open("200_senate.txt","w")  #the file containing the final generated senate format


out.write("MAT.NUMBER\tNAMES\tMTH201\tGP\t MTH202\tGP\tMTH203\tGP\tMTH204\tGP\t
MTH205\tGP\tMTH207\tGP\tMTH209\tGP\n")

for line in file :

        result = “”                          #empty at the beginning
```

```
line = line.rstrip()

line = line.split('\t')

p = line[1]  #matnumber

q = line[2]

result = p +'\t'+ q


if p in MTH201 :     #if p in MTH201 meaning candidate took the course

result = result +  "\t" + MTH201[p]

else :               #candidate did not take the course

result = result +  "\t" + " +"\t" + "


if p in MTH202 :

result = result +  "\t" + MTH202[p]

else :               #candidate did not take the course

result = result +  "\t" + " +"\t" + "


if p in MTH203 :

result = result +  "\t" + MTH203[p]

else :               #candidate did not take the course

result = result +  "\t" + " +"\t" + "


if p in MTH204 :

result = result +  "\t" + MTH204[p]

else :               #candidate did not take the course

result = result +  "\t" + " +"\t" + "


if p in MTH205 :

result = result +  "\t" + MTH205[p]

else :               #candidate did not take the course

result = result +  "\t" + " +"\t" + "
```

```
        if p in MTH207 :
        result = result +  "\t" + MTH207[p]
        else :              #candidate did not take the course
        result = result +  "\t" + '' +"\t" + ''

        if p in MTH209 :
        result = result +  "\t" + MTH209[p]
        else :              #candidate did not take the course
        result = result +  "\t" + '' +"\t" + ''

        out.write(result +'\n')

out.close()
```