Antoine Gicquel
29/11/2020

M2-SIF
Final report - DSL

In this report, I will describe the CSVs (CSV script, source extension is csvs) project which aims to facilitate CSV file manipulation.

I – Implementation details

I was alone to develop the project. In consequence, only two variants were made : python and bash. However, the DSL was intended to be use with dynamically typed languages. The python variant have more features than the bash variant. I was not able (in time) to make an interpreter.
We can see the differences in the following table :

	Python	**Bash**
Load Command	Fully	Fully
Store Command	Fully	Fully
Create Command	Fully	Fully
Set Command	Fully	Mostly
Print Command	Fully	Fully
Export Command	Fully	Fully
Delete Command	Partially	Partially
Add Command	Fully	Fully
Rename Command	Fully	Fully
Apply Command	Partially	None
Merge Command	Fully	None
Concatenation Command	Fully	Fully
While Command	Fully	Mostly
If Command	Fully	Mostly
Comparison Expression	Fully	Mostly
Arithmetic Expression	Fully	Mostly
Count Expression	Fully	Fully
Selector Expression	Mostly	Partially
Field Selection Expression	Mostly	Partially
Last Expression	Fully	Fully
Variable Manipulation	Fully	Partially

II – Variants comparison

In order to compare the two variants, I created multiple test files with features that are implemented in both variants and with CSV files with different length. The purpose of those files is to check the correctness of the results and the resources usage (with */usr/bin/time* the command).
To make the assessments, a script shell will : 1) compile the test files for the two target; 2) execute the resulting programs; 3) store results and resources usage; 4) create a CSV report based on the previous files.

The following figure is the CSV report file exportation :

Test name	Execution type	Execution time	CPU usage	Max memory (kbytes)	Result equivalent
/sources/calcul_float_10000	py	0:00.57	71%	61572	different
/sources/calcul_float_10000	sh	2:30.63	104%	43848	different
/sources/calcul_float_100	py	0:00.29	98%	56848	different
/sources/calcul_float_100	sh	0:00.68	109%	3816	different
/sources/calcul_float_5000	py	0:00.33	99%	59572	different
/sources/calcul_float_5000	sh	0:54.79	106%	23044	different
/sources/calcul_int_10000	py	0:00.36	99%	61972	same
/sources/calcul_int_10000	sh	2:29.83	104%	43892	same
/sources/calcul_int_100	py	0:00.28	98%	56620	same
/sources/calcul_int_100	sh	0:00.69	110%	3760	same
/sources/calcul_int_5000	py	0:00.32	98%	59508	same
/sources/calcul_int_5000	sh	0:54.77	106%	23128	same
/sources/concat_10000	py	0:00.34	97%	63740	different
/sources/concat_10000	sh	1:42.52	114%	74620	different
/sources/concat_100	py	0:00.31	99%	57196	different
/sources/concat_100	sh	0:00.53	136%	4392	different
/sources/concat_5000	py	0:00.32	98%	61236	different
/sources/concat_5000	sh	0:35.45	116%	38452	different
/sources/print_10000	py	0:00.31	99%	62120	different
/sources/print_10000	sh	0:36.40	114%	43892	different
/sources/print_100	py	0:00.30	99%	57652	different
/sources/print_100	sh	0:00.19	121%	4068	different
/sources/print_5000	py	0:00.30	98%	59500	different
/sources/print_5000	sh	0:13.38	116%	22932	different
/sources/print_col_float_10000	py	0:00.37	99%	61948	different
/sources/print_col_float_10000	sh	1:49.55	107%	43892	different
/sources/print_col_float_100	py	0:00.28	98%	56808	different
/sources/print_col_float_100	sh	0:00.61	111%	3808	different
/sources/print_col_float_5000	py	0:00.33	99%	59984	different
/sources/print_col_float_5000	sh	0:43.13	108%	22992	different
/sources/print_col_int_10000	py	0:00.38	99%	61932	same
/sources/print_col_int_10000	sh	1:49.45	107%	43884	same
/sources/print_col_int_100	py	0:00.28	98%	56588	same
/sources/print_col_int_100	sh	0:00.62	111%	3816	same
/sources/print_col_int_5000	py	0:00.33	99%	59632	same
/sources/print_col_int_5000	sh	0:42.97	108%	23036	same

- Resources usage:
  The bash variant is much slower than the python one. Furthermore, it has a bigger CPU usage. The python is better but use reasonably more memory.
- Correctness:
  In the test scenario, we evaluate a strict equality of the outputs. However, most of the outputs are equivalents. In float operation, the python variant add extra zero after the last digit. And for the printing, the python variant add extra table embellishments.

III – Conclusion

To answer to the question “What is the best variant”, the obvious response, based on the previous statements and assessments, is the Python variant. It performs much better than the Bash variant on every criteria. Despite the most recent features of Bash as *associate array and regex*, it was hard to imitate the Python core functions and the dynamic typing. This is due to the DSL design and with the experience acquired through the project, I would design differently if I had to do it all over again. Moreover, during the project, only the AST was used. An intermediate representation like a three-address code would allow us a static type check and optimizations.

The Eclipse project and the test files are available at the following link :
https://github.com/agicquel/csvs