FULangS: A Capstone Scripting Tool

Francisca O. Oladipo, Memunat A. Ibrahim, Abdul-Ahad U. Obansa, Abdulwahab A. Jatto

***Abstract*— This paper reports the development experiences and features of FULangS, a quasi, general-purpose scripting paradigm developed at the Federal University Lokoja, Nigeria as part of the requirements for a capstone course in Survey and Organization of Programming Languages during the 2016/2017 Academic Session. FULangS was written in C programming language, with the Flex/lex environment for lexical analysis, Bison/yacc for semantic analysis and Cygwin, to build and compile the flex/lex and yacc/bison files. The language is compiled for the Microsoft Windows environment, and in addition to being functional, FULangS is also an imperative computer scripting paradigm, possessing the ability to describe a sequence of steps that change the state of the computer. FULangS scripts interprets to a virtual machine and offers special feature support for stack machines and garbage collection.**

***Index Terms*— capstone, programming paradigm, scripting, quasi-environments.**

I. INTRODUCTION

Generally, Programming describes a process that starts from the original construction of a formal solution to a computing problem and leading to the creation of codes based on the syntax and semantics of a language –a programming language. A programming language is an instruction tool that structurally turns a formulated solution to a formalized one based on its syntax and meaningful combination of its elements [1], [2].

Computer languages are often classified according to their purpose(s) of use. In this vein, programming languages refer to those class of computer languages used by programmers to develop software programs, scripts, or other sets of instructions for computers to execute. Other classifications are: markup languages, style sheet languages and scripting languages. The author also grouped some languages based on the development type: applications and programs development, artificial intelligence development, for database development, game development, computer drivers or other hardware interface development, internet and web page development, and script development. In addition to the above classifications, author's preference is also a major factor in grouping computer languages and more specifically, we have the imperative paradigm which is the oldest and executes its commands sequentially; the object oriented (OO) paradigm which represents software tokens in terms of attributes/variables/parameters and behaviours/methods/state. Other specific classes are functional and logical paradigms-both of are at close proximities to the problem domain and can solve complex problems.

A computer Algorithm or computer program consists of two essential parts, a description of actions which are to be performed and a description of data which are manipulated by these actions. Actions are described by statements and data are described by declarations and definitions. The data are represented by values of variables. Every variable occurring in a statement must be introduced by a variable declaration, data types are used to define the set of values which may be assumed by the variable.

This paper reports on the evolution of a quasi-scripting language named FULangS from a capstone undergraduate course at the Federal University Lokoja Nigeria during the 2016/2017 session. A detailed description of the scripting paradigm is presented in the next sections in addition to the language elements and sample codes for the FULangS tool.

A. Scripting Programming Paradigm

Scripting is the art of writing scripts in between programs written in programming languages like C, java, and between scripts written in another scripting language. *Script codes are sequence of instructions and are often interpreted and the primitive elements are in the form of direct calls for elementary tasks and API. They can be embedded into large component-based software to perform complex in critical-sensitive applications*, deployed to write codes that target and automate operations on a software system. A summary of the description of the scripting paradigm is given by [3] as "a programming language that is used to manipulate, customize, and automate the facilities of an existing system. In such systems, useful functionality is already available through a user interface, and the scripting language is a mechanism for exposing that functionality to program control".

Scripting is essential as it allows a quicker development of applications than the conventional method; encouraging a rapid modification of applications as their requirements change [4]. Unlike programming language, scripting languages has been found very useful in developing new applications from 'off the shelf' components and act as a glue connecting existing software components to create a new one [5]. Scripting languages are very useful in developing web pages [6], software applications, embedded systems [7], operating system shells, and games.

Scripting languages are most suitable in experimental programming where efficiency is not as important as the development speed to rapidly develop and test experimental software or prototypes [8]. They are also used in the back-end to control and support applications with programmable front-end, and in developing a dynamic and responsive web pages.

Francisca O. Oladipo, Computer Science Department, Federal University, Lokoja, Nigeria, +2348034725167
Memunat A. Ibrahim, Computer Science Department, Federal University, Lokoja, Nigeria
Abdul-Ahad U. Obansa, Computer Science Department, Federal University, Lokoja, Nigeria
Abdulwahab A. Jatto, Computer Science Department, Federal University, Lokoja, Nigeria

B. *Characteristics of scripting languages*

1) Scripting languages are easier to learn and work with: their simple syntax, abstraction of memory management and data storage makes them to be easily accessible even to non-programmers. This feature also makes them to be preferred when teaching beginner programmers [9].
2) Easier to debug: applications written in scripting languages are easier to debug since interpreter's report errors as they are encountered and the program execution is paused, making the location and removal of errors easier. Since errors are reported immediately, beginners can write programs with less difficulty [10].
3) Low overheads and ease of use: scripting languages increases the productivity of the users with features they provide. Also, scripting languages provides powerful string manipulation (regular expressions), and easy access to low-level kernel services [6].

C. *The General Anatomy of a Scripting Language*

Most scripting languages codes like php, javascript begin with an opening tag and end with a closing tag while others like python have no tags. This is followed by preprocess instructions like import statements. Unlike traditional programming languages, scripting languages have no entry point or main function, just block of statements or expressions. These expressions are written by combining the language elements which could be numbers (in base 10), strings (in double quotation marks), arithmetic operators, bitwise operators, relational operators, logical operators. The elements are combined to form mathematical expressions, call expressions which apply a function to a set of arguments, Input/output Expressions, control structures expressions. Also, every scripting language provides its user with a means of writing comments as a form of documentation.

II. Review of Related Works

Various scripting languages has been built for different purposes; for example: tool control language (TCL) was designed to be linked with C libraries, to control C programs; they are used for scripting embedded systems. Facilities built on TCL, Expect for example allows easy scripting of interactive programs unintended to be controlled by a script, while TK allows a very fast designing of dialog box, buttons, etc. It is very easy and flexible, with a consistent syntax and has higher economy of expression than c, java, etc [10]. The major limitation of TCL is its weak modularity and namespace control facilities, making writing of big programs hard. TCL is implemented for both Unix and Windows Operating System.

Practical Extraction and Report Language (Perl) is a scripting language majorly popular for string processing, system administration, and web maintenance [11]. It was developed by Larry Wall to replace AWK and to be used Unix script programming. Perl more advanced language than shell as it has stronger data types. Perl decreases the usage of C for applications with little requirement for performance or memory optimization. The drawbacks of Perl include the difficulty in reading and understanding its making it difficult to learn. Also, maintaining Perl codes as it becomes bigger is also strenuous.

Python is a scripting language with a simple syntax that is in between the Modula and C family. It can be integrated in other application or used alone [9]. It can be integrated with C and can import from and export data to C libraries that are loaded dynamically. Its code block structure is controlled by indentation. It provides both the object-oriented programming and the procedural programming style or a mixture of both styles. Python is suitable for writing dynamic web pages, it can also use the Tk toolkit to build GUIs making it suitable for building robots and network administration scripts. Python and Java are the only programming languages for large projects with multiple developers. Python is simpler and friendlier than Java, it also support rapid prototyping making it the most popularly taught programming language [12]. Python are implemented both for Microsoft and Macintosh operating systems.

Lisp is a functional language with lists and trees as its main data types. Lisp automatically manages memory, and operators overloading and garbage collection was first introduced in it [13]. The major drawback of Lisp is the fact that it is a collection of languages and not a single language, thereby preventing it from delivering what it promised. Currently, it is most suitable for tasks involving interactive processing of database or file, building applications that are integrated with text editors, or text editing functionalities. An extension of Lisp is LISp-Miner Control Language which is a language designed to automate the data mining process and provides a means to control the features of LISp-Miner system [7].

Job Control Language (JCL) was for controlling programs and tasks, Visual Basic for programming Microsoft office Applications, Markup Languages for distinguishing structure from content, and making documents interactive, Practical Extraction and Report Language (Perl) for generating reports and creating dynamic web pages, Python language for accessing kernel services of the Amoeba operating system [14].

III. Materials and Methods

FULangS is written in C as the parent language with other tools for compiler construction. FLEX (Fast Lexical Analyzer Generator) was deployed as lexical analysis and YACC (Yet Another Compiler Compiler) was used as parse generator. Other tools used in the development of FULangS are:

1) Cygwin x32, x64 -a Microsoft Windows platform for GNU utilities (with all the necessary packages for compiling lex and yacc)
2) Windows operating systems (both X32 and X64 bit)

IV. The FULangS Paradigm

Generally, a computer programming language must have four elements which are primitives (built-in things), means of combination, means of capturing common patterns, and means of abstraction. FULangS has three elements out of the four elements mentioned above which are primitives (built-in things), means of combination, and means of abstraction. Primitives consist of primitive expressions and statements, representing the smallest or simplest element of the language. Examples are numbers (in base 10), strings

(in double quotation marks), arithmetic operators, bitwise operators, relational operators, logical operators.

Means of combination: defines means by which simpler elements are combined to form compound ones (expressions). Expression types can be:

a) Mathematical expressions which uses infix notation.
b) Call expressions which apply a function to a set of arguments. The call expression consists of the operator (function name) and the operands (arguments list) which are enclosed in parentheses. The operands can be primitive elements, mathematical expressions or call expressions.
c) Input/output Expressions: to accept input, the keyword 'fulin' is used while 'fulout' is used to print output.
d) Containment: control structures which are specified using if, for loop, while, do while statements are used to segment a block of codes and specify when they are executed.

Means of abstraction: defines means by which the elements are named and used as a unit.

1) Naming of identifiers: the assignment statement is used for naming values, it consists of a name to the left of = and a value to the right. = is the assignment operator.
2) Data abstraction: data structure is used to specify the data type of an expression. The data structures implemented in FulangS are integer, double, string.

A. *Language Elements*

This section described the vocabularies of the FULangS scripting language.

1) Keywords

Table 1 lists the set of terms with special meaning in FULangS

Table 1. *FULangS* Keywords

Keywords	Meaning/Functions
Fulprog	*Begins the program*
Fulvar	*Variable declaration*
End	*Ends the program*
Or	*Logical Or operation*
And	*And*
If	*If*
For	For
Do	Do
Skip	Skip
<>	Comment single line and multi-line
While	While
Fulin	Input
Fulout	Ouput
Else	Else
Endif	End if
Endfor	End for
Endwhile	End while

2) Comments

Comments are like helping text in the *FULangS* program and they are ignored during execution of the program. They start with < and terminate with the character > as shown below:

```
<MyFirstProgram>
```

3) Delimiters

In a *FULangS* program, the semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity. Given below are two different statements:

```
Fulout "Hello World";Fulin x;
```

4) Identifiers

A *FULangS* identifier is a name used to identify a variable item. An identifier starts with a letter A to Z or a to z followed by zero or more letters, underscores, and digits (0 to 9). *FULangS* does not allow punctuation characters such as @, $, and % within identifiers and they are case-sensitive. Thus, 'Power' and 'power' are two different identifiers in *FULangS*. Here are some examples of acceptable identifiers:

```
mohdzaraabcmove_name
a_123
myname50
Temp
 j
a23b9
retVal
```

5) Variables

A variable is a name given to a storage area that *FULangS* programs can manipulate. Variable in *FULangS* has a specific type Integer, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable. The name of a variable can be composed of letters, digits, and the underscore character but it must begin with either a letter or an underscore. As earlier pointed out, upper and lowercase letters are distinct because variable names in *FULangS* are case-sensitive. Variable list may consist of one or more identifier names separated by commas and terminated by semicolon. Example:

```
Fulprog
  i,j,k,l;
  fulvar
  <OtherStatements>;
       end
```

6) Operators

FULangS operators and their description are listed below:

+	Adds two operands
−	Subtracts second operand from the first
*	Multiplies both operands
/	Divides numerator by denominator
%	Modulus Operator and remainder of after an integer division
=	Checks if the values of two operands are equal or not, if yes, then condition becomes true.
<>	Checks if the values of two operands are equal or not, if values are NOT equal, then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if YES, then condition becomes true.

<	Checks if the value of left operand is less than the value of right operand, if YES, then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes, then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes, then condition becomes true.
<<	left shift
>>	Right shift
|	Bitwise Or
&	And
#	Exclusive Or
OR	Or
AND	And

7) Expression Types

Expressions are constructs denoting rules of computation for obtaining values of variables and generating new values by the application of operators. *FULangS* expressions could be made up of a single element or combination of multiple elements and can be used to specify arithmetic operations, bitwise operations, relational operations, logical operations and string manipulation.

a. Assignment expression is used to assign strings or numbers or an expression to a variable.
b. Arithmetic expressions are used to perform arithmetic operations on numbers and strings.
c. Relational expressions are used to compare two expressions to return true or false.
d. Logical expressions are used to combine two or more relational expressions and also return true or false.
e. Function call expressions are used to call in-built functions.

Below are the different type of *FULangS* expression with examples (Table 2).

8) Data Types

The current release of FULangS accepts only integers, string and double as basic datatypes. All other types are derived from these three.

9) Functions

A function is a group of statements that together perform a specific task. The FULangS language provides two sets of built-in functions: math functions (Table 3) and string functions (Table 4).

Table 3. FULangS Built-in Math Functions

Math function	Description
`mathSqrt()`	This function is used to find square root of the argument passed.
`mathCeil()`	This function returns nearest integer value which is greater than or equal to the argument passed.
`mathFloor()`	This function returns the nearest integer which is less than or equal to the argument passed.
`mathFabs()`	This function returns the absolute value of the argument passed.
`mathExp()`	This function is used to calculate the exponential "e" to the x^{th} power.
`mathLog10()`	This function is used to calculates base 10 logarithm.
`mathLog()`	This function is used to calculates natural logarithm.
`mathSin()`	This function is used to calculate sine value.
`mathCos()`	This function is used to calculate cosine value.
`mathTan()`	This function is used to calculate sine tangent.
`mathSinh()`	This function is used to calculate hyperbolic sine.
`mathCosh()`	This function is used to calculate hyperbolic cosine.
`mathTanh()`	This function is used to calculate hyperbolic tangent.
`mathAtan()`	This function is used to find arc tangent.
`mathAsin()`	This function is used to find arc sine.
`mathAcos()`	This function is used to find arc cosine.
`mathMod()`	This function is used to find modulo arithmetic.
`mathPow()`	This function is used to find the power of the given number.
`average()`	This function returns the average value of the arguments passed.
`min()`	This function returns the average value of the arguments passed.
`max()`	This function returns the average value of the arguments passed.
`sum()`	This function returns the average value of the arguments passed.
`mul()`	This function returns the average value of the arguments passed.

10) FULangS Special Features

Garbage Collection

FULangS uses a modified mark-sweep algorithm to implement the Boehm-Demers-Weiser conservative garbage collector. FULangS allows programmers allocate memory normally, and automatically recycles memory when it determines that it can no longer be otherwise accessed without causing the programmer to explicitly de-allocate the memory when no longer in use.

Conceptually it operates roughly in four phases, which are performed occasionally as part of a memory allocation which are the Preparation phase, Mark phase, Sweep phase and Finalization phase.

Virtual machine

The FULangS virtual machine is employed to allow the implementation of control structures in the language. Interpreting if, for loop, while, and do while control structures was not possible, as a result virtual machine was employed to compile the control structures.

Stack Machine

Stack machine are implemented in FULangS to evaluate arithmetic operations, bitwise operation, input and output and terminate the program execution.

Sample Programs

The "Hello World"

The FULangS implementation of the "Hello World" is shown below (Figure 1)

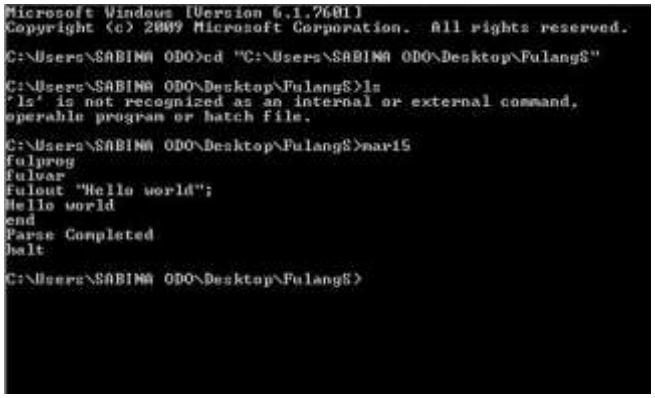


Figure 1. *FULangS* implementation of the "Hello World" script

Other sample scripts are listed below:

```
==================================================
=
<ProgramPrintEvenNumbers>
fulprog
na,x;
fulvar
fulinna;
fulin x;
whilena =< x do fuloutna; na = na+2; endwhile;
end
==================================================
=
<programTestwhile>
fulprog
na,x;
fulvar
fulinna;
fulin x;
whilena =< x do fuloutna;na++; endwhile;
end
==================================================
=<FibonacciSeriesProgram>
fulprog
prev,next,sum,n,count;
fulvar
fulin count;
n = 1;
prev = 1;
next = 1;
while n =< count do fuloutprev; sum = prev + next;prev
= next; next = sum; n = n+1; endwhile;
end
==================================================
=
<programSumNumbers>
fulprog
i,sum;
fulvar
sum = 0; i=1;
while i =< 1000 do sum = sum + i; i = i + 1; endwhile;
fulout sum;
end
==================================================
=
<programShortIncrement>
fulprog
s;
fulvar
s=3;
s=s++;
fulout s;
end
==================================================
=
<programBitwiseOperation>
fulprog
fulvar
fulout 3 | 4;
fulout 3 & 4;
fulout 3 # 4;
fulout 3 << 4; fulout 3 >> 4;
end
==================================================
=
<programShortSub>
fulprog
x;
fulvar
x = 2;
fulout x-=2;
fulout x;
end
==================================================
=
<programArithmeticTest>
fulprog
na,x;
fulvar
fulinna;
fulin x;
fulout na + x;
fulout na * x;
fulout na - x;
fulout  na / x;
fulout na % x;
fulout na^3;
end
==================================================
=
fulprog
fulvar
fulout "HelloWorld";
end
==================================================
=
<programtofactorial>
fulprog
fac,i,num;
fulvar
fulinnum;
fac = 1; i=1;
for i =<num do fac =fac * i; i=i+1; endfor;
fuloutfac;
end
==================================================
=
<programtShortArithmetic>
fulprog
a,b;
fulvar
a = 4; b = 2;
fulout a += b;
fulout a *= b;
fulout a -= b;
fulout a /= b;
fulout a %= b;
end
==================================================
=
<programprintnumbers>
fulprog
i,x;
fulvar
fulin x;
i=0;
for i =< x do fulout i; i = i+1; endfor;
end
==================================================
=
<programTestConditions>
fulprog
a,b,c;
fulvar
fulin a; fulin b; fulin c;
if (a<b) or (b<c) then
fulout (a+b)*c; else
fulout (a*c); endif;
if (a>b) and (b<c) then fulout (a*c);
else fulout (a+c); endif;
end
==================================================
=
<programAp>
fulprog
i,sum,x;
fulvar
fulin x;
```

```
sum=0; i=1;
while i=<x do sum=sum+i;
i=i+1; endwhile;
fulout sum;
end
```

```
=================================================
=
fulprog
prev,nex,sum,n,coun;
fulvar
fulin coun;
n = 1;
prev = 1;
nex = 1;
while n =< coun do fulout prev; sum = prev + nex;prev
= nex; nex = sum; n = n+1; endwhile;
end
```

```
=================================================
=
<programPrintSum>
fulprog
i,sum;
fulvar
sum = 0;
i=1;
while i =< 1000 do sum = sum + i; i = i + 1; endwhile;
fulout sum;
end
```

```
=================================================
=
fulprog
s;
fulvar
s=3;
s=s++;
fulout s;
end
```

```
=================================================
=
fulprog
x;
fulvar
x = 2;
fulout x-=2;
fulout x;
end
```

```
=================================================
=
fulprog
na,x;
fulvar
<testOperators>
fulin na;
fulin x;
fulout na + x;
fulout na * x;
fulout na - x;
fulout na / x;
fulout na % x;
fulout na^3;
end
```

```
=================================================
=<programExpressions>
fulprog
a,b;
fulvar
a = 4; b = 2;
fulout a += b;
fulout a *= b;
fulout a -= b;
fulout a /= b;
fulout a %= b;
end
```

```
=================================================
=
<programFnctions>
fulprog
fulvar
fulout average(4,5,6,7,8,2);
fulout sum(5,9,6,45,5,3,4,65,7);
fulout mul(2,56,8,5,3,5);
fulout min(5,7,5,43,9);
fulout max(23,68,7,3,3,6,3,56);
end
```

```
=================================================
=
<programFunctions>
fulprog
x,y,z,;
fulvar
x = average(4,5,6,7,8,2);
y = sum(5,9,6,45,5,3,4,65,7);
z = mul(2,56,8,5,3,5);
w = min(5,7,5,43,9);
t = max(23,68,7,3,3,6,3,56);
fulout x;
fulout y;
fulout z;
fulout w;
fulout t;
end
```

```
=================================================
=
<StringTest>
fulprog
next,prev;
fulvar
next = strlen("Abdulwahab");
prev = strlen("Obansa");
if next == prev then fulout next; else fulout prev;
endif;
end
```

```
=================================================
=
<StringTest>
fulprog
fulvar
fulout concat("Memu","natu");
fulout isEqual("Memunat","Memunatu");
fulout strlen("Memunat");
fulout toUpper("Memunat");
fulout toLower("MEMUNATU");
fulout revstr("Menunat");
end
```

```
=================================================
=
<LogicalOperators>
fulprog
fulvar
fulout 3 | 4;
fulout 3 & 4;
fulout 3 # 4;
fulout 3 << 4; fulout 3 >> 4;
end
```

V. Conclusion

The quasi-scripting language *FulangS* has been designed based on the flex and bison working on the Cygwin environment to facilitate the construction of verifiable scripts. The development of the programing language *FULangS* is based on two principal aims, first to make available a set of syntax suitable to teach scripting as a systematic discipline based on certain fundamental concepts, and also to develop a set of semantic dimensions which are both efficient and reliable on presently available PC.

FULangS can be used for command line scripting i.e. the codes can run without server, browser or GUI (Figure 2). It can also be used to perform arithmetic operation, strings manipulation and developing small applications and only accepts procedural codes. The alphabets of the language are similar to those of other quasi-scripting paradigms but FULangS provides dual supports for both compilation and interpreting. The compiled part of *FULangS* supports only integer arithmetic operations while the interpreted part supports floating-point arithmetic, integer arithmetic and few string operations as it only accepts only integers, string and double as basic datatypes in its current release. Other datatypes however can be specified in terms of these three.

FULangS support implicit garbage collection using the Boehm-Demers-Weiser approach.

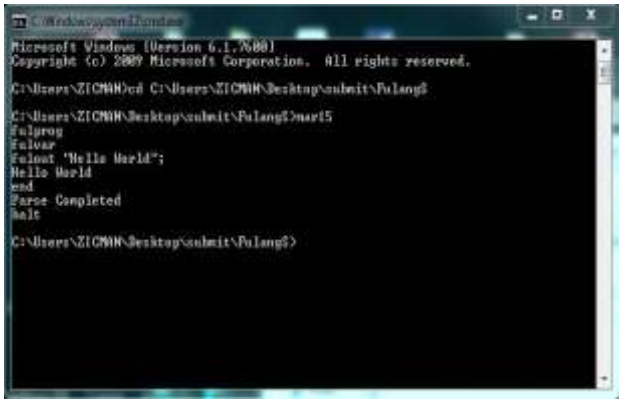


Figure 2. Command line scripting in *FULangS*

REFERENCES

[1] Bebbington, S. (2014a). "What is coding". Blog post by Shau Bebbington on the Yearofcode project. Accessed July 2017 from http://yearofcodes.tumblr.com/what-is-coding

[2] Bebbington, S. (2014b). "What is programming". Blogpost by Shau Bebbington on the Yearofcode project. Accessed July 2017 http://yearofcodes.tumblr.com/what-is-programming

[3] Schwalb, E. M. (2004). ITV Handbook: Technologies and Standards. Prentice Hall, New Jersey, USA. Pp. 222

[4] Richards, G., Lebresne, S., Burg, B., & Vitek, J. (2010). An Analysis of the Dynamic Behavior of JavaScript Programs. PLDI'10. Toronto, Ontario, Canada.: ACM.

[5] Thiemann, P. (2008). Types and Analysis for Scripting Languages. Mini Lecture, Tallinn, Estland Universitat Freiburg, Germany. Retrieved 09 01, 2017, from IoC: cs.ioc.ee/tarmo/tasl08/tasl/pdf

[6] Patra, P. K., & Pradhan, P. L. (2014). Dynamic Virtual Programming Optimizing the Risk on Operating System. Telkomnika Indonesian Journal of Electrical Engineering12 (8), 6369-6379.

[7] Simunek, M. (2014). LISp-Miner Control Language description of scripting language implementation. Journal of Systems Integration 2014 (2), 28-44.

[8] Ray, B., Posnett, D., Filkov, V., & Devanbu, P. (2014). A Large Scale Study of Programming Languages and Code Quality in Github. FSE'14. Hong Kong: ACM.

[9] Gu, L., Yan, N., & Xiu, Y. (2017). Discussion on Teaching Methods and Choice of Programming Language on Software Engineering Major. 2017 Asia-Pacific Engineering and Technology Conference (APETC 2017) (pp. 1681-1686). Kuala Lumpur: DEStech Publications, Inc.

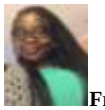
[10] Nanz, S., & Furia, C. A. (2015). A Comparative Study of Programming Languages in Rosetta Code. ICSE '15 Proceedings of the 37th International Conference on Software Engineering - Volume 1. Pages 778-788

[11] Doug, S. (2000). "Beginner's Introduction to Perl". Retrieved 10-07-2017 from https://www.perl.com/pub/2000/10/begperl1.html

[12] Godwin-Jones, R. (2015). The evolving roles of language teachers: Trained coders, local researchers, global citizens. Language Learning & Technology, 19(1) pp. 10–22

[13] David, C. (2011). Influential Programming Languages, Part 4: Lisp. Downloaded 10-09-2017 from www.informit.com/articles/article.aspx?p=1671639

[14] Kanavin A. (2002). An overview of Scripting Languages. Lappeenranta University of Technology. Finland, December, 2002. pp.10



Francisca O. Oladipo is an Associate Professor and current Head, Department of Computer Science, Federal University, Lokoja Nigeria.

Memunat A. Ibrahim is completing her B.Sc program in Computer Science at the Federal University Lokoja, Nigeria. She is passionate about engineering of programming languages and programming tools to make learning easier.

Abdul-Ahad U. Obansa,b is a final year student in the Department of Computer Science, Federal University Lokoja, Nigeria. Ahad's research interests is in development of usable tools for humanity.

Abdulwahab A. Jatto is a final year student in the Department of Computer Science, Federal University Lokoja, Nigeria. His main research focus is development of scripting paradigms for low-scale hardware.

Table 2. *FULangS* Expression Types

Expression	Sign	Description	Expression Description	Example
Assignment expression	=	Assign strings or numbers or an expression to a variable	identifier=Expression ;	`A=5; B=A+2;`
Arithmetic expression	+ / - * %	Addition Division Subtraction Multiplication Modulus	Expression+ Expression; Expression/ Expression; Expression- Expression; Expression* Expression; Expression% Expression;	`A+2; A/B; A-B; A*B; A%B;`
Relational expression	== != > < >= <=	Is equal to Is not equal to Is greater than Is less than Is greater than or equal to Is less than or equal to	Expression==Expression Expression!=Expression Expression>Expression Expression<Expression Expression>=Expression Expression<=Expression	`A==B A!=B A>B A<B A>=B A<=B`
Logical expression	OR AND	Or And	Expression or Expression Expression and Expression	`A>B or B>C A>B and B>C`
Function call expression			Expression (Expression)	`mathFloor(3)`