The jan Programming Language

Jan C. Bierowiec

1 Introduction

There are many libraries used to simulate, animate, and visualize calculations. In JavaScript, p5.js is used for simulations/animations on the web-browser. In C++ gnuplot is used for visualizations made for calculations. In Python, the Manim library is used to create STEM-related animations for better understanding. The idea here is that, for different programming languages, different libraries need to be created and used in order to achieve a desired result for calculation, visualization, simulation, and animation. I therefore, propose to create a new language whose sole purpose would be do exactly those things.

2 Design Goals

2.1 Main Goals

When designing this language, the main goal was for this language to be used by students, researchers, scientists, and programmers interested in any form of calculation, visualization, simulation, or animation computations. This language would facilitate implementation for domain experts and those interested in this field of computer science, while still being an introductory programming language that anyone can learn without a significant learning curve.

2.2 Evaluation Criteria

Ideally the language would be optimal in all evaluation criteria, those being: readability, writability, reliability and cost.

- Readability:
  - The language would have detailed documentation, similar to that of Python, where if a programmer does not know how to solve a problem, he/she can look it up, and the full documentation would be provided.
  - When looking at a program, the user would have a good understanding of the code from top-down via the use of indentation.

  - Methods of forming compound statements and identifiers would be implemented so that the programmer and user know what each part of the code is referring to.

- Writability:
  - The language would be easy to write which would avoid errors, making the program more likely to be correct.
  - This would bring the learning curve cost down of the language as anyone can easily pick up on the language.
  - The language would support abstraction so that the code does not have to be replicated in all places where needed.

- Reliability:
  - The language would be strictly related to computing and developing calculations, visualizations, simulations, and animations.
  - The language would have type checking provided by the compiler.

- Cost:
  - The language would have a smaller learning curve as it would be catered to a broad audience of students as well as a wide audience of developers and scientists interested in calculations, visualizations, simulations, and animations.
  - With the language being high on reliability, the language should have lower costs.

2.3 Other Design Considerations

The language would also be high functioning, with the intention that memory would be allocated automatically. This is important as an automatic memory management system (garbage collection) would provide memory safety, thus reducing opportunities for memory leaks. This came into consideration with respect to visualizing data and performing calculations on large sets of data.

3 Targeted Domains

3.1 Institutional Domains

Scientific:
This domain was considered with the intention that scientists, developers, and researchers can easily make large calculations in this language and plot their results to easily visualize the work they are doing.

Educational:

This domain was considered with the intention to have this language be an introduction to young developers and to have them make a connection between the pen and paper mathematics taught in school, and directly applying it to the computer language interpreter to help better visualize the mathematical operations at play.

3.2 Technical Domains

Real-time:
The language would be real-time in order for large datasets as an example, to be computed and processed quickly. Having a real-time language would also be useful due to the fact that results from written code would be predictable.

Computationally intensive:
The language would also be computationally intensive, similar to a language like C++ in order to make fast calculations and other additional actions based on coder's choice.

4 Targeted Users

Domain experts:
Scientists and researchers interested in creating simulations for otherwise, complex mathematical computations, that take a while to run in different languages, not always resulting in proper results. This language would be useful as it would save time.

Students:
As a language that is not only object-oriented, but also procedural, this language would be a good introduction for students to understand how the subjects of mathematics and computer science are interdisciplinary. It would also be a beneficial language for students to learn as it would provide a great start for students interested in generating plots, developing animations and simulations as these are skills that can be transferred over to the domains of Data Science or Game Development.

Professional programmers
The language would also be used by professional programmers in a variety of fields, but most definitely it would help the large scope of Data Scientists as well as programmers who would want to play around with and are interested in developing visualizations for a wide range of topics.

5 Type of Language

The language would be a *compiled language* as compiled languages are have faster execution time than *interpreted languages* and since the goal of the lan-

guage is to calculate, simulate and visualize mathematical calculations correctly and accurately, this is why this is considered.

The language would also be both *procedural* and *object-oriented*. The purpose of this is for the language to be catered towards a broad as well as niche audience. The broad audience, would use the procedural method so that the language can be learned from a top-down approach, making learning the language useful for students with a low learning curve. The language would also be object-oriented, which would allow for code to be reusable through inheritance. This would be useful for programmers and scientists working on similar problems but with different requirements or parameters. Along with that, being an object-oriented language it would be easily scalable for more complex computations.

6 Features

The language features would definitely involve a new syntax, with a mix of different things from other languages. The language would look like C++ for the most part, however instead of using ';' as the ending of each line of code, the '.' would be used instead. Also with respect to input and output, such as:

```
cout << ... << endl;
```

or

```
cin >> value;
```

It would be simply replaced by:

```
output()
```

and

```
input()
```

Which is similar to Python's `print()` and `input()` functions respectively. This would benefit the language in making it easily readable & writable. To define functions, the

```
f()
```

will be used, which is different from:

```
def()
```

or

```
int main()
```

which is used in Python or declaring the function data type in C++ respectively.

The language along with being a compiled language, would also have an interactive debugger that would let the programmer visualize the code. It would use more of the visual cortex, graphs, and visualizations. This would be useful for both students and scientists alike as not only would the programs compile quickly, but when compiling the program, they would see the logic behind the program and what to fix if the debugger runs into an error allowing for easy backtracking.

The language would also support concurrency and parallelism. The main idea behind this is for scientists specifically, who might want to run multiple programs with different parameters without having to run one program at a time. Not only would this improve the functionality of the CPU, but it would allow for efficient performing of tasks.

7 Inspiration

7.1 Idea:

The inspiration for this language was based on personal experience when trying to research ways to simulate physics formulas and have students relate to what they are learning in the classroom with the real world. When performing this research, p5.js was chosen, which is a JavaScript library that makes creating simulations relatively easy with a lot of the syntax looking similar to that of C++. Moving forward however, there was a want and desire to develop more aesthetic simulations with real components, similar to the work of 3blue1brown.

7.2 Language References

C++ as a programming language is known to be fast because it is a compiled language, so when theorizing a new language, this fact and capability needed to be preserved. This feature makes executing problems take less time than a program with the same semantics, but takes more time, such as in the case of Python.

Some of Python's simplicity would be borrowed however, such as some of Python's built in statement functions like:

```
print()
```

in order to make printing values simpler.

For the language, I would also implement Java's strong exception handling mechanism to secure memory in order to prevent memory leaks (which is something that C++ does not have) and so that it prevents you from continuing running the program until the error is corrected. The language would also have access modifiers and specifiers such as private, protected, public and default. This would help in controlling the access of any member.

8 Example Program

Below is a sample program of computing numbers for the $y = x^2$ mathematical function. A function is declared to compute the square of a number, and then that function is called in the main program code. The values are then printed, using Python-like syntax.

```
f sq(x):
    return x * x.

let i.
let x.

output("Enter a number").
input(x).

output("All of the squares up until", x, "are: ").
for(i = 1. i <= x. i = i + 1.):
    let result = sq(x).

    output(result, ", ").
```

9 Syntax

Using Backus-Naur Form (BNF), below is how the language is formally defined:

```
<program> ::= start <statement_list> end

<statement_list> ::= <statement> | <statement> ; <
    statement_list>

<statement> ::= <declaration>
              | <definition>
              | <expression>
              | <control_statement>
```

```
                | <function_call>
                | <function_declaration>
                | <return_statement>
                | <output_statement>
                | <input_statement>

<function_declaration> ::= f <identifier> ( <parameter_list
    > ): <statement_list>
<parameter_list> ::= ( ) | ( <identifier> ) | ( <identifier
    > , <identifier> )

<return_statement> ::= return <expression> .
<output_statement> ::= output ( <expression_list> ) .
<input_statement> ::= input ( <expression_list> ) .

<expression_list> ::= <expression> | <expression> , <
    expression_list>
<expression> ::= <literal>
                 | <identifier>
                 | <function_call>
                 | ( <expression> )
                 | <expression> <operator> <expression>

<declaration> ::= let <identifier> .
<definition> ::= let <identifier> = <expression> .

<control_statement> ::= <if_statement>
                      | <for_loop>
                      | <do_while_loop>

<if_statement> ::= if ( <expression> ) { <statement_list> }
<for_loop> ::= for ( <declaration> . <condition> . <
    expression> ) { <statement_list> }
<do_while_loop> ::= do { <statement_list> } while ( <
    expression> ) .

<condition> ::= <identifier> <operator> <expression>

<function_call> ::= <identifier> ( <optional_args> )
<optional_args> ::= <expression> | <expression> , <
    optional_args>

<literal> ::= <numeric_literal> | <string_literal> | <
    array_literal>
<string_literal> ::= ' <characters> '
<numeric_literal> ::= <digit> | <digit> <numeric_literal>
<array_literal> ::= [ <numeric_literal> , <numeric_literal>
     ]
```

```
<identifier> ::= <letter> | <identifier> <letter> | <
    identifier> <digit> | <identifier> _
<letter> ::= a | b | ... | z | A | ... | Z
<digit> ::= 0 | 1 | ... | 9
<operator> ::= + | - | * | / | ^ | <= | >= | < | > | =
```