Background

As I learned LISP, I was amazed by its language design. However, LISP is far away from a c-family programming language and is unfriendly to most programmers. LISP requires a strong mathematical support and a short term memory to decode its expression far away from human habit. Therefore, when I used LISP, I could do nothing but to design algorithms. For instance, defining a new variable is similar to use a for function or do as simple as an addition operation. LISP is more elegant than pragmatic.

On the other hand, LISP's essence, functional programming, is admirable. Although expressions are hard to read, it has a clear structure. If some operations such as defining new variables and evaluating mathematical expressions are made easier, functional programming can be much useful.

My most favorable realization of functional programming is in Django. Django is a web server framework written in Python. Its design to retrieve data from databases is creative and is very "functional programming". It proves that given a competent community support and a competent library such as Python, functional programming is powerful.

Therefore, I decide to experiment how powerful combining the design of functional programming with a language similar to C.

Language Design

The Set Programming Language is designed to use mathematical sets as a fundamental data structure in the programming language. Sets has two unique properties: the numbers don't overlap and sets are unordered. Inspired by python and LISP, and with the help of sets, the iteration process can be simplified, and the Set Programming Language is able to be designed as a functional programming language, where the specific keywords are designed as functions, and the Set Programming Language features no keyword.

Lambda Expression

The **lambda** expression

```
>> square = {x:x*x}
```

denotes the function in c

```
float square(float x){
  return x*x;
}
```

The separator ':' separates variable declaration and result calculation (expressions). Programmers can use ';' to separate different expressions. This design is to enable functions with multiple expressions. The Set Programming Language does not feature a standard definition for a function. Therefore, adding the ';' operation can supply more complicated functions when needed.

```
>> multiply = {x,y:a=x*y;a}
```

has the same effect as the lambda function before. In this case, it is equivalent to the following code segment in c

```
float multiply(float x, float y) {
    float a = x * y;
    return a;
}
```

For Loops (Map)

For loop in Set Programming Language is similar to map in python or LISP, but has a name of `for` to show that this function is powerful enough to replace for loops. In python, the expression `for a in b: f(a)` is equivalent to a for loop in Set Programming Language with code `for(b, f)`

In detail, the following for loop in c

```
for (i=0;i<10;i+=2){
  printf("%d\n", i);
}
```

can be expressed in SET by

```
>> for(range(1,10,2),println)
1
3
5
7
9
```

The range function is similar to function, it is used as

```
>> range(initial,stopping,interval)
```

and its output is a set

As shown, for loop is a function, the first parameter is a set, and the second is a function. For loop will take every value in the set as the only parameter of the function, it returns the set containing all the return values from the function.

For example,

```
>> print(for(range(1,10),{x:x*x}))
{1, 4, 9, 16, 25, 36, 49, 64, 81}
```

Similar to for-loops

```
>> if(1==2/2, {print("TRUE")}, {print("FALSE")})

TRUE
```

In this case, if the first condition is true, then {print("TRUE")} will be executed, otherwise {print("FALSE")} is executed.

If a lambda function has no parameter, ':' mark to se can be omitted.

Implementation

The source code of Set Programming Language is published in

https://github.com/2000jedi/Set-Programming-Language/tree/master/Go

The interpreter is built with Go, a programming language designed by Google to take the place of C.

In the source code

- lex.go is the lexical analysis. It takes in the original code and parse it into several lexical, such as '=' is stored as OPERATOR and '"TRUE"' is stored as a STRING
- syn.go is the syntactic analysis. It converts the results in lexical analysis and produce the Reverse Polish Notation, which is easier for the interpreter to run.
- execute.go is the interpreter, it takes in the results in syn.go and execute them.
- ds_*.go are data structures, including array, set, function, string, and numbers. The number calculation is not stored according to IEEE 754 standard, but as the form $\frac{\text{numerator}}{\text{denominator}}$ for maximize precision to do scientific work.
- builtin.go contains all the built-in functions by default contains for, range, if, import, print, println, and exit. These functions are crucial expressions in other languages, but dealt as functions to realize a truly functional programming language with no keyword.

Future

Reduce is another core function for functional programming. Currently, the for function process a set separately, but lacks a function to deal with the set in general. Reduce function achieves this.

Whether to add `class` or `struct`, feature of object-oriented programming, is debatable. In one hand, it destroys elegance in functional programming and makes the Set Programming Language more similar to python. In another hand, without some OOP feature, the convenience and ability of Set Programming Language is limited.

A competent function library is crucial to a programming language. With supporting functions such as I/O processing and standard math library, the Set Programming Language could said to be complete.