I. Introduction

sPy is an interpreted language that has the basic facilities and functionalities similar to Python. Some of the constructs (especially the keywords) are in Spanish.

1. Elementary Operations
    - Arithmetic operations (+, -, *, /, %)
    - Relational operators (<, <=, >, >=, ==, !=)
    - Logical operators (||, &&)
    - Unary operator (!, -, +)
    - Assignment operator (=)
2. Control Statements
    - si
    - otro
    - mientras
3. Input/Output
    - introducir
    - imprimir

II. Lexical Rules of sPY (lexeme & tokens)

Token Name	Lexeme	Type
start	comienzo	reserved keyword
if	si	keyword
else	otro	keyword
while	mientras	keyword
do	hacer	keyword
return	regresso	reserved keyword
print	imprimir	reserved keyword
scan	introducir	reserved keyword
for	para	keyword
call	avisar	keyword
is	es	keyword
try	tratar	keyword
end	fin	reserved keyword
break	romperse	reserved keyword
class	clase	reserved keyword
def	explicar	reserved keyword
end_def	finexplicar	reserved keyword
LT	<	relational operator
GT	>	relational operator
LEQ	<=	relational operator
GEQ	>=	relational operator
EQ	doubleEqual	relational operator
NEQ	!=	relational operator
NOT	!	boolean operator
OR	||	boolean operator
AND	&&	boolean operator

STRLIT	*string*	string literal
CHRLIT	*char*	character literal
INTLIT	int	numeric literal
FLTLIT	float	numeric literal
DBLLIT	double	numeric literal
boolean	boolean	numeric literal
PLUS	+	arithmetic operator
MINUS	-	arithmetic operator
MUL	*	arithmetic operator
DIV	/	arithmetic operator
MOD	%	arithmetic operator
OPENPAR	(	grouping symbol
CLOSEPAR	)	grouping symbol
OPENCURLY	{	grouping symbol
CLOSECURLY	}	grouping symbol
OPENBRACE	[	grouping symbol
CLOSEBRACE	]	grouping symbol
ASSIGN	equalSign	assignment operator
COMMA	,	punctuation
TILDE	~	punctuation

III. Syntax of sPy Language

<mainProgram> → **comienzo** {(statements)~} **fin**
| **comienzo** <classes> **fin**
| **comienzo** <functionHeading> **fin**

<classes> → **clase**{<identifierList> <optionalStatements>}

<functionHeading> → **explicar** ({parameters~}):

{<statement/s>~}
fin_explicar

<identifierList> → (A | B | C | D | … | Z) <identifierList>
| (a | b | c | d | … | z) <identifierList>
| (0 | 1 | 2 | … | 9) <identifierList>
| _
| ε

<type> → **int** <identifierList>~
| **float** <identifierList>~
| **double** <identifierList>~
| **char** <identifierList>~
| **string** <identifierList>~

<optionalStatements> → <statementList>
| ε

<statementList> → <statements>
| <statementList> <statements>

<statements> → <identifierList> = <expression>
| **introducir** <type>
| **imprimir** (<identifierList>)
| **imprimir** "<identifierList>"

| **regresso**(<expressionList>)
| <optionalStatements>
| **si** (<expressionList>)
<statements>
| **si** (<expressionList>)
<statements>
| <comment>

<elseIf>
<elseIf> → **otro** (<expression>)
<statements>
<elseIf>
| ε

<while> → **mientras** (<expressionList>) ~
<optionalStatements> ~
| **mientras** (<expressionList>) ~
<optionalStatements> ~
Romperse

<expression> → <expression>

| <expressionList>, <expression>
| <identifierLists>

<expressionList> → <logicalExpression>
| <arithmeticExpression>

<logicalExpression> → <expression> **<** <expression>
| <expression> **>** <expression>
| <expression> **==** <expression>
| <expression> **<=** <expression>
| <expression> **>=** <expression>
| <expression> **!=** <expression>

<arithmeticExpression> → <expression>
| <expression> **+** <expression>
| <expression> **–** <expression>
| <expression> ***** <expression>
| <expression> **/** <expression>

<comment> → ** <identifierList> ~ <comment>
| *~ <identifierList> *~ <comment>
| ε

IV. Typing Rules

i. Declarations

For the sPy declaration typing rule, the data type must be declared first by the user. The syntax should go like this:

<data_type> <variable_name>~

If you wish to initialize a value for the variable, it will go like this:

<data_type> <variable_name> = <initial_value>~

Here are some notes you need to remember in declaring variables:

The spacing between the data type and the variable name is important

The tilde (~) after the variable name as well.

This is a case sensitive language so if you declare a variable with a lowercase letter/s, it must be consistent throughout the whole code.

To access a variable, it must be consistent with the declaration

Declaration can be done anywhere in the program as long as a variable **MUST** be declared first before it can be used or accessed.

Spacing between the tilde (~) and the variable name/initial value is optional as well as the equal sign(=) between the variable name and the initial value

Variables must begin with a letter and may be followed only by letters, numbers and underscores.

ii. Type Consistency Requirements

A type determines the set of values and operations specific to values of that type.

The following are the variable types of sPy (lower-case letters required):

Float	float
Integer	int
Character	char
String	string
Boolean	boolean

iii. Function Definitions

Declaring a function in sPy must be started with the reserved keyword explicar (all in lower-case letters) and must be ended with finexplicar.

Syntax goes like this:

explicar <function name> (param: <parameters>):
    <statement/s>~
fin_explicar

Here are some notes you need to remember in declaring functions:

The passed parameters must be placed too inside the parenthesis after the "param:" reserved keyword

Data types of the variables must be consistent and what the function needs

Calling:

If you wish to call the defined function, the syntax is:

avisar <function_name>(param:<parameters>)~

Note that before a function can be called, it must be first declared and specified. They can be anywhere in the code as long as it's placed before the call statement.

Returning:

If you wish to return a value to the main program, returning expressions from the function is accomplished through:

regresso (<expression>)~

A returned value can be assigned to a variable as specified below:
<variable_name> = avisar <function name>(param:<parameters>)~

iv. Expressions

Expression are usually created to produce a new value or it specifies the computation of a value by applying operators and functions to operands. There are no strict typing rules for expressions in sPy.

a. **Operands** are the objects that are manipulated and operators are the symbols that represent specific actions. For example, in the expression. 5 + x. x and 5 are operands and + is an operator.
All expressions **have at least one** operand. An operand may be a literal, a constant, a variable or function.

b. **Arithmetic Expressions** - Mathematical operators and functions will be called with this syntax:
<expression1><operator><expression2>

An expression containing an operator with infinite number of arguments can be expressed with the following syntax and can be used with parentheses "( )"

<expr1><operator><expr2>((<operator>) <expr2>)(<operator>)<expr3> . . .)

Level of Precedence: in an expression with multiple number of arguments, the precedence of operators will still follow the PEMDAS rule. Expressions inside a parenthesis will be computed first, then the ones with exponent, then multiplication, division, addition, subtraction.

c. **Boolean Expressions** – Boolean expression are expressions that results in a Boolean value, that is, in a value of either true or false. There are only few boolean operators but more complex boolean expressions can be built out of simpler expressions. Here is the list of the Boolean operators available for sPy with their meanings:

Operator	Name	What it means
&&	And	True if and only if both sides are true
||	Or	True if either one side is True (or both are True)
!	Not	Changes true to false and False to True

There are six arithmetic tests that can be used to create boolean values:

Operator	Name of operator
<	less than
<=	less than or equal to
==	equal to
!=	not equal to
>=	greater than or equal to
>	greater than

These different operators can be used in this syntax:

<arg1> <boolean_operator> <arg2>

Levels of Precedence: in an expression with multiple number of arguments, the precedence of operators will simply go from left to right. Left operators will be analyzed and so on until the last operator. Syntax goes like this:

<arg1> <boolean_operator1> <arg2> <boolean_operator2> <arg3> <boolean_operator4> <arg4><boolean_operator5><arg6>…<argn>

V. Operational Characteristics

a. Data - Variables can store data of different types, and different data types can do different things. Here are the different data types available for sPy and how we deal with each of them:

Integer – with the keyword int, will be used for mathematical operations. int should be specified before a variable (where value will be stored) followed by equal sign (=) then the statement/ integer value. A variable with data type int can be initialized with a value or not.

Float – float is a data type that is used as well in mathematical operations just like the int, everything is just the same as the int except that float can provide

precision of 6 to 7 decimal digits. Just simply type the keyword float before the float value.

Double – a double data type is more precise than float. A double variable can provide precision up to 15 to 16 decimal points.

Characters – with the keyword char, it is used to return a character value given an integer value. Keyword char followed by an integer value

String – with the keyword string, it is used to declare a sequences of character data or just a character data such as words or alphabets. Strings in sPy can have spaces and can be enclosed in either single quotes (') or double quotes (" - the double quote character), or three of the same separate quote characters ('' or """).

Below is an example of declaring variables with different data types:

```
comienzo
    int a = 0 ~
    float b = 0.00 ~
    double c = 1.23 ~
    char 3 ~
    string “hello world” ~
fin
```

b. Arithmetic Expressions – refer to Chapter IV b. for a more detailed explanation. Here’s an example for a simple addition of two variables:

```
comienzo
    int a = 9 ~
    int b = a+5 ~
fin
```

c. Boolean Expressions - refer to Chapter IV c.

d. Assignment Statements - assignment statement sets and/or re-sets the value stored in the storage location(s) denoted by a variable name; in other words, it copies a value into the variable. In most imperative programming languages, the assignment statement (or expression) is a fundamental construct. It is fundamental because it makes data accessing easier and more efficient and lets the programmer keep track of the data. The syntax on assignment statements can be seen in chapter 4 declarations and is applicable to every data type available in sPy.

VI. FUNCTIONS

To improve understandability and re-usability sPY is divided into building blocks called functions. It is a set of instruction that is used to perform a single, related action to avoid rewriting the same logic. sPY function contains 3 aspects, namely:

Function Call

It is use to invoke an actual function. The function is either built-in or user defined. Calling a function can be done many times and from any part of the sPY program.

Function Definition

It is use to provide the required functionality of a declared function. This contains all the statements that are to be executed in a sPY program.

Function Declaration

It is done to inform the compiler about the function name, function parameters and return values data type in a sPy program.

Example:

```
**function declaration
explicar square ( float x ):
** main function, program starts from here
```

```
comienzo

   float m, n ~
   imprimir ( "Enter some number for finding square ")~
   introducir (m) ~
   ** function call
            n = square ( m ) ~
            imprimir ( "Square of the given number is",m,n )~

explicar square ( float x )   ** function definition
            float p ~
            p = x * x ~
            regresso ( p ) ~ **return statement
fin
```

VII. PROGRAM EXECUTION

a. Program Start and Termination

A function that indicates the starting point of a program execution and its termination. A function is led by a term comienzo and ended with a term fin

The syntax is as follow:

```
comienzo
        <statement> ~
        <statement> ~
        <statement> ~
fin
```

b. Comments

Inline comments in Spy start with the symbol ** (two asterisk) until the end, and multiline comments start with *~ end with ~* (asterisk tilde) and vice versa.

Example 1:

```
**this is a comment in one line
```

Example 2:

```
*~ this is a comment
on more lines ~*
```

c. Expression Statement

A statement is usually made up of an expression. In sPy an expression statement is simply an expression followed by a ~ (tilde). It has either calling of functions and methods or receiving operations.

Example:

```
a = 0 ~          ** an expression statement a = a + 1 ~
**an expression statement imprimir("Hello World") ~
**an expression statement
```

d. Conditional Statement

A conditional statement is a set of rules performed if a certain condition is met. In Spy a conditional statement start with reserved keyword si in where a condition is met and a reserved keyword otro in where an action is perform.

The syntax is as follow:

```
si (<boolean expression>)
        <statement/s>~ **will execute if boolean expression is true
otro
        <statement/s>~ **will execute if boolean expression is false
```

e. Loop Statement

Loop statements in sPy start with a reserved keyword para. The reserved keyword romperse is for terminating the loop and the sPy program resumes at the next statement following the loop.

The syntax is as follow (without termination):

```
para ( statement1~statement2~statement~)
        **code block to be executed
```

(with termination):

```
para ( statement1~statement2~statement~)
        <condition(s)> ~
romperse        **terminates the loop when condition is false
```

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

Another looping in sPy has a reserved keyword hacer and mientras. It is to execute the code block once, before checking if the condition is true, and then it will repeat the loop as long as the condition is true.

Example:

```
int i = 0~
 hacer
    imprimir(i)~
    i++~

   mientras (i < 5)~  **condition
```

Output:

0 1 2 3 4