Nova Lang

General Syntax:

In Nova language, each statement must end with a semicolon(;), it can be a declaration of a variable, conditional statements, loops, print, and so on. Moreover, throughout the documentation, if something like <xyz> is encountered, then it is a placeholder where xyz specifies what it should be.

Types:

Nova lang would support three types, namely numbers, booleans, and string. The numbers are defined as int, booleans as bool, and strings as string. Further, the number could be extended to float to store floating point numbers such as 1.04, but as of now, we have not included float data type.

Integer	→	int
Booleans	→	bool
String	→	string

Bool Type:

There are only two types- true, false

Variable Declaration:

Variables in Nova can be declared using the **var** keyword followed by the declaration of the type(int, string, bool) and its name. One thing to note: it is also necessary to initialise the variable, and once a variable is declared, it cannot be redeclared in any of the scopes(same or child). By default, all the variables in Nova are mutable, but only the same type of values can be assigned if needed; otherwise, it will throw an error.

var <type> <variable_name> = <value>;

e.g., var int count = 5;

Strings:

Strings are defined using double quotes(“here is an example of stings”). In Nova, string supports concatenation, slicing, and indexing.

Indexing:

Indexing can be used to access characters in a string from a particular position. Like most of the programming languages, index in Nova would also start from 0,i.e., 0-based indexing.

```
var string text = “Hello, World!”;
println(text[7]);
```

Let's break it out:

H e l l o , W o r l d !

0 1 2 …

Therefore, the output would be ‘W’ without quotation marks.

Slicing:

The slicing operation takes a string and returns a slice of it. This takes two indices, the start and the end index and the end index is excluded.

```
var string text = “Hello, World!”;
println(text[1 : 3])
```

This would output “el” without quotation marks.

Concatenation:

Two or more strings can be concatenated using pipe (|) operator.

e.g.,

```
var string s1 = “Tea”;
var string s2 = “Post”;
var string s3 = s1 | s2;
println(s3);
```

The output will be “TeaPost”.

Arrays:

Arrays can be thought of as a collection of homogenous data types. In Nova, it can be declared in the same way as a variable using square brackets.

```
var array temp = [5, 7, 3, 2];
        Or
var array(4) temp;
```

Thus, there are two ways to declare an array one is to explicitly write each of the values, and the other is to specify the size which would result in initializing with 0’s.

It supports four operations:

length : *temp.length* returns the length of the sequence

head : *temp.head* returns the first element of the sequence

tail : *temp.tail* returns the last element of the sequence

cons() : *temp.cons(element)* adds the element to the head of the sequence

e.g.

```
var array temp = [5, 7, 3, 2];
println(temp.length);
println(temp.head);
println(temp.tail);
temp.cons(22);
println(temp.head);
```

Output:

4
5
2
22

Indexing:

Indexing in arrays is similar to strings. The syntax is as follows-

var array temp = [1, 4, 6, 2, 7, 9];
println(temp[4]);

O/P- 7

. ,

Tuples:

Tuples are similar to arrays but with two constraints- these are immutable, and once declared, the size can't be changed and that said, we have to explicitly put the values.

var tuple temp = (5, 7, 3, 2);

Operators:

Arithmetic Operators:

Nova Lang supports four basic types of arithmetic operations: addition, subtraction, multiplication, and division. All these operations can be performed on numbers only.

Addition → +
Subtraction → -
Multiplication → *
Division → /

The above operators are for binary operations, - can be used as unary operators. The below table summarizes the usage.

Operator	Example	Explanation
+	expr + expr	Arithmetic addition
-	expr - expr	Arithmetic subtraction
*	expr * expr	Arithmetic multiplication
/	expr / expr	Arithmetic division
-	-expr	Returns oppositely signed operand.
!	!expr	Reverses the logical state of the operand

We would also like to add a few more unary operators such as post-increment(exp++) and post-decrement(exp--). Both of these would either increase or decrease the value of the operand by 1 in the next line; that is, if the value of an int data type, let's say count, is post-incremented, then in the same line, its value would not change, but from the next line onwards it would be treated as count + 1. But this is an idea which might be implemented.

Assignment Operator:

There is only one assignment operator in Nova, i.e., = .

```
var int demo = 7;
```

Here, the value of 7 is assigned to the demo variable which is of int type.

Comparison Operators:

Nova supports six types of comparison operators: >, <, >=, <=, !=, ==

Logical Operators:

Operator	Example	Explanation
and	expr and expr	Logical and operation on two boolean operands
or	expr or expr	Logical or operation on two boolean operands
not	not expr	Logical negation of boolean operand

Control Flow:

If-else statements:

Branching with if-else is similar to other languages. The condition should be surrounded by parentheses, and the code that needs to be executed inside should be inside curly braces, i.e., a block statement.

```
if(condition) {
        //code block
};
else {
        //code block
};
```

If the condition inside the *if* block turns out to be true then only the *if* code block gets executed. Otherwise, only *else* code block gets executed.

Loop:

In most of the languages, there is support for 2 types of loops *while* and *for.* In Nova, there is only one with the keyword **loop** and it works like a while loop in most of the languages. The code block inside will be looped till the condition stays true.

```
loop(condition) {
        // block of code
};
```

e.g.,

```
var int x = 0;
loop ( x < 5 ) {
        x = x + 1;
};
```

Print Statement:

A print statement in Nova would look like this, where println stands for “print line”.

```
println(“Hello, World!”);
```

Function:

To define any function in Nova Language, the syntax is as follows:

```
fn <function_name>(parameters)  {
        //function body
        return <value>;
};
```

Closures:

```
fn outer (){
        var string outervar = “Hello”;
        fn inner (){
                println(outervar);
        };
};
inner();
```

This prints out “Hello” even though function inner has no access of outervar, showing closure in the Nova language.

Exception-handling:

The basic structure of exception handling in Nova involves the use of the ‘try’ and ‘catch’ blocks. The ‘try’ block encloses the code that may potentially throw an exception, and the ‘catch’ block defines the code to handle the thrown exception.

Syntax for try-catch:

```
try {
        // code that may throw an exception
};
catch(errorType) {
        // handling division by zero error
        println("Error: Division by zero");
};
catch(e) {
        //Default block for other errorTypes
};
```

Syntax for throw:

```
throw errorType;
```