Backus Naur Form (BNF)

A language is regular if it can be represented by a regular expression and an FSM.

However, some languages cannot be expressed using regular expressions.

A context-free language like Backus-Naur form (BNF) is necessary whenever an infinite number of elements need to be counted.

All regular languages can be represented by context-free languages.

Production Rules
BNF is expressed using production rules. For instance, a bit is defined with the following production rule.

```
<bit> ::= 0 | 1
```

This means that a bit can take on the value 0 or 1.

- < > is a non-terminal symbol. In this example `<bit>` is a non-terminal symbol and 0 and 1 are terminal symbols.
- ::= states that the right-hand side is defined by the left hand side
- | means a choice (OR) between two symbols

Non-Terminal Symbols
If there is a non-terminal symbol on the right hand side there should be another production rule with the non-terminal symbol on the left.

This is demonstrated in the following example of three production rules where the non-terminal symbols <month> and <year> appear on both the left and right-hand side.

```
<date> ::= <month>/<year>
<year> ::= 2018 | 2019 | 2020
<month> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 11 | 12
```

Valid expressions include 2/2020 or 12/2019

Recursion
Recursion allows us to have one or more of a symbol. Consider the following example:

```
<integer>::=<digit>|<digit><digit>|<digit><digit><digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 |7 | 8 | 9
```

These pair of production rules only allow us to express a maximum value for an integer 999. Of course integers can have an infinite number of digits. We represent this using recursive BNF production rules.

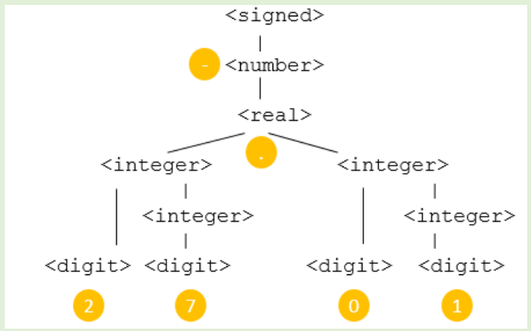
```
<integer> ::= <digit> | <digit><integer>
```

As we know with recursive functions we need a base case and general case. The base case is `<digit>` on its own and the general case is `<digit><integer>`

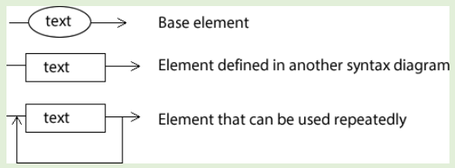
Parse Tree
We are going to extend our rules so that we can define real numbers and negative numbers.

```
<signed> ::= <number> | +<number> |-<number>
<number> ::= <real> | <integer>
<real> ::= <integer> . <integer>
<integer> ::= <digit> | <digit><integer>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 |7 | 8 | 9
```

Let us check that -27.01 is a valid expression for these production rules by using a parse tree.



Syntax diagrams
Syntax diagrams are another way of expressing the syntax of a language. The symbols for the syntax diagram are:



The syntax diagram for the BNF production rules that we have look at is given as: