A JQ program consists of one or more combined expressions that operate with JSON values and produce zero or more JSON values.

JSON values

object
  {}
  { *members* **}**
members
  pair
  pair **,** *members*
pair
  string **:** *value*
array
  []
  [*elements* **]**
	elements
	  value
	  value **,** *elements*
	value
	  string
	  number
	  object
	  array
	  true
	  false
	  null
		string
		  ""
		  " *chars* **"**
		chars
		  char
		  char chars
		char
		  any Unicode character except **"** *or* ****
		    or control character
		  \\" \\\\ \\/
		  \\b \\f \\n \\r \\t \\u*four-hex-digits*
		number
		  int
		  int frac
		  int exp
		  int frac exp
			int
			  digit
			  digit1-9 digits
			  - *digit*
			  - *digit1-9 digits*
			frac
			  . *digits*
			exp
			  e digits
			digits
			  digit
			  digit digits
			e
			  e e+ e- E E+ E-

JSON values can be bound to variable names with the **as** construct. Besides the constants **null**, **false**, **true** and number and string literals, values can be defined with the following constructs:

Value (de)constructors

Syntax	Description
(…)	scope delimiter and grouping operator
[…]	array constructor; collects generators output
{…}	object constructor, with extended syntax in relation to JSON
term as *pattern*	binding of variables with array and object destructuring

New expressions are built with operators and special constructs. In increasig order of priority the operators are:

Operators

Operator	Assoc.	Description
|	right	connects two filters
,	left	produces one value after another
//	right	alternative value for `null`, `false` or `empty`
= |= += -= *= /= %= //=	nonassoc	assign, update; `a @= b` == `a = a @ b`
or	left	boolean "or"
and	left	boolean "and"
!= == < > <= >=	nonassoc	boolean tests
+ -	left	polymorphic plus and minus; prefix negation
* / %	left	polymorphic multiply, divide; modulo
?	none	coerces errors to the empty value

Evaluation flow is organized with the operators *pipe* and *comma* and the constructs **if**, **reduce**, **foreach**, **label** and **try**. The postfix *question* operator is syntactic sugar for the **try** construct.

The basic syntax for all JQ special constructs is as follows:

Programming constructs

```
def name: expression;
def name(parameters): expression;
term as pattern | expression
if expression then expression else expression end
if expression then expr elif expr then expr ... else expr end
reduce term as pattern (init; update)              # init, update and extract
foreach term as pattern (init; update)             # are expressions
foreach term as pattern (init; update; extract)
label $name | expression ... break $name
try expression
try expression catch expression
```

Filters receive one input value and zero or more parameters, and produce zero or more output values; new filters, with function-like syntax, can be defined with the **def** construct.

Core predefined filters

Filter	Description
.	produces unchanged its input value; it is the *identity* filter
.k ."k"	object member access; shorthand for .["k"]
x[k]	array element and object member access
x[i:j]	array or string slice
[]	generate values in objects and arrays
..	equivalent to: ., .[]?, (.[]?|.[]?), (.[]?|.[]?|.[]?), ...
keys	generates ordered array indices and object keys
empty	filter that does not produce any value on its output
error, error(value)	signals an error
length	size of strings, arrays and objects; absolute value of numbers
del(path)	remove path from the input value
type	name of JSON values type
explode, implode	string to/from code point array conversion
tojson, fromjson	JSON value to/from string conversion
@fmt "\(expr)"	format string and string interpolation

Two important predefined filters are *dot*, the filter that does nothing, and *empty*, the filter that never produces values. The main laws for those filters and the *pipe* and *comma* operators are:

Laws for *dot*, *empty*, *pipe* and *comma*

. | a ≡ a a | . ≡ a	empty , a ≡ a a , empty ≡ a
empty | a ≡ empty a | empty ≡ empty	a , (b , c) ≡ (a , b) , c (a , b) | c ≡ (a | c) , (b | c)