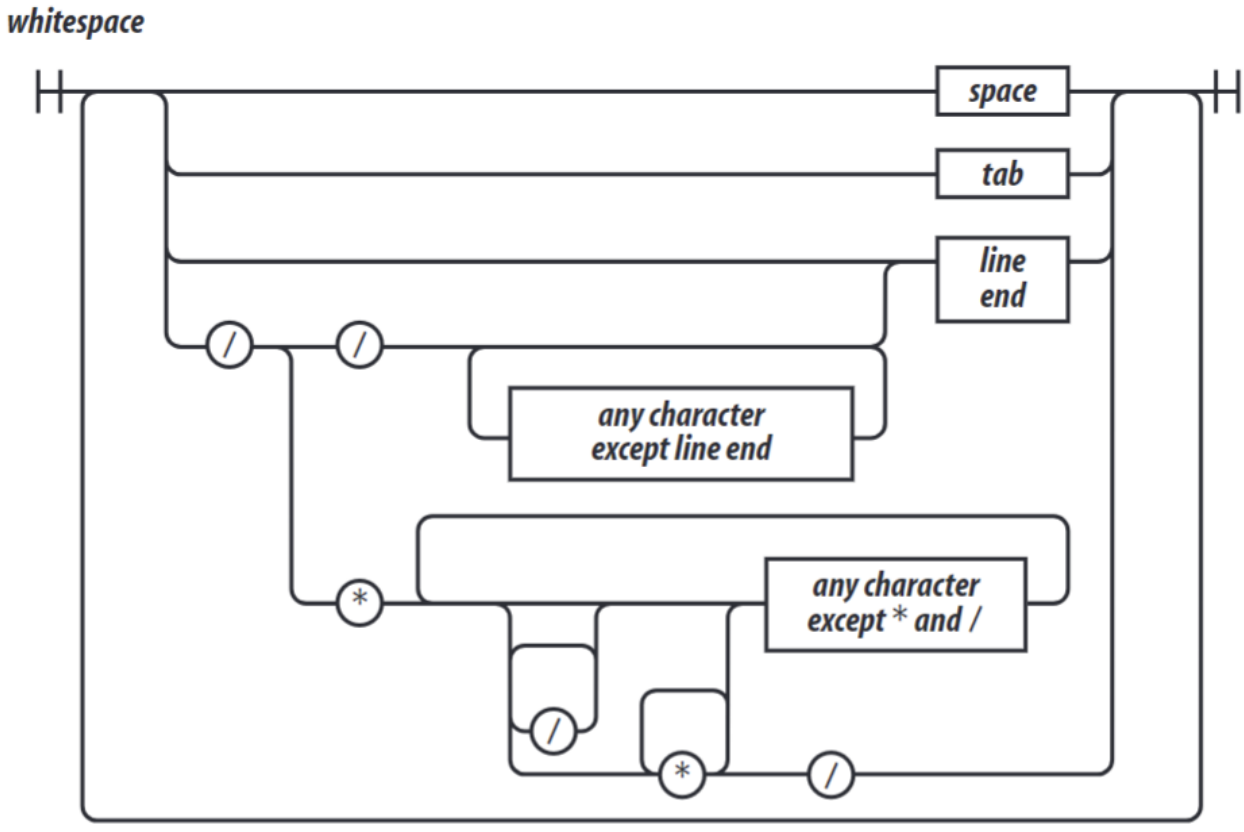
Grammar

This section will largely show how JavaScript is structured as a language. It will pull diagrams from JavaScript: The Good Parts.

This note will largely show how JavaScript is structured as a language. It will pull diagrams from JavaScript: The Good Parts.

Whitespace

Whitespace is any section of your code that doesn't have any characters present. It is sometimes necessary to separate certain tokens, and sometimes not necessary. That being said white space makes code a lot more readable



Comments

JS has 2 forms of comments.
Block comments are formed by `/* */` where anything in the middle is a comment
Line-ending comments are formed with `//` where the comment terminates at the end of the line

---

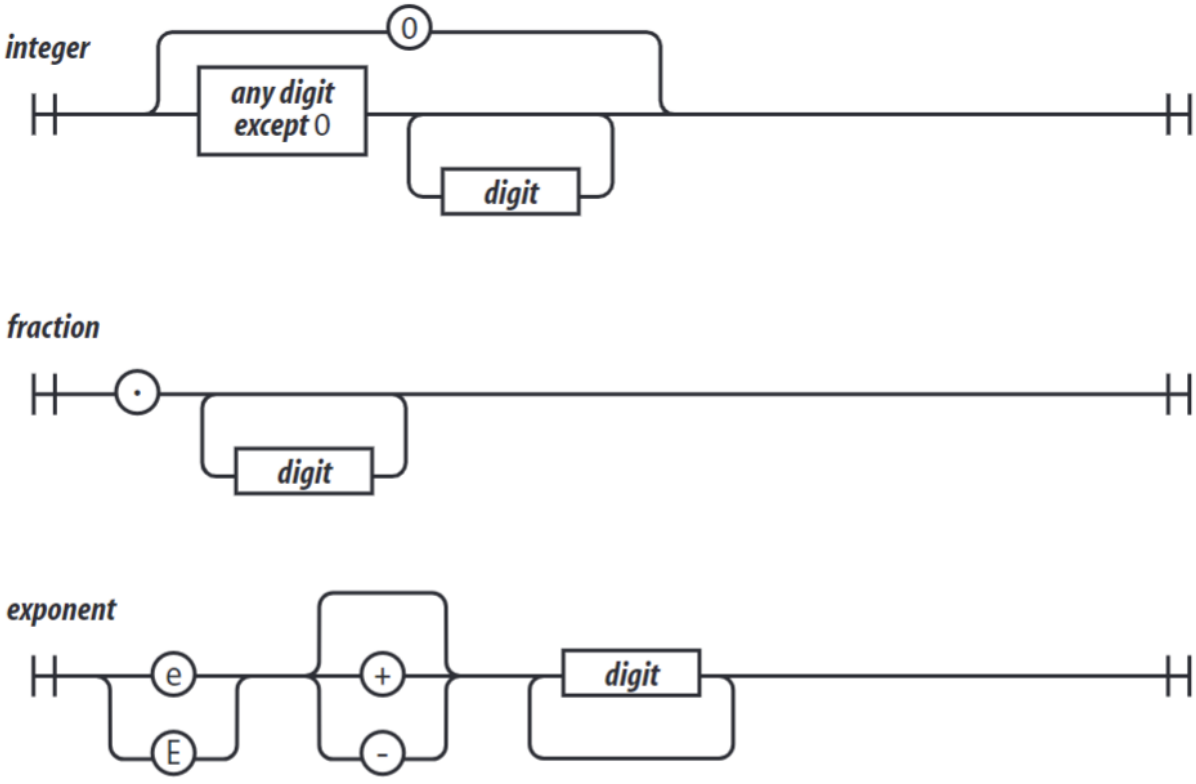
Names

Naming things always start with a letter and then can optionally be followed with any number of letters numbers or underscores; Except for any language keywords of which there are many (I'm not listing them lol)

Names are used for statements, variables, parameters, property names, operators and labels.

Numbers

JavaScript only has a single number type. a 64-bit floating point (same as java's double). There is also no separate Integer and floating point types, so `1` and `1.0` are the same value (thank god).



Negative numbers can be formed by the `-` prefix operator

`NaN`

`NaN` or not-a-number is returned whenever a numerical operation cannot return a numeric result. Typically means something has gone wrong. `NaN` is not equal to anything, including itself, so the only way to check for `NaN` is using the `isNaN(number)` function

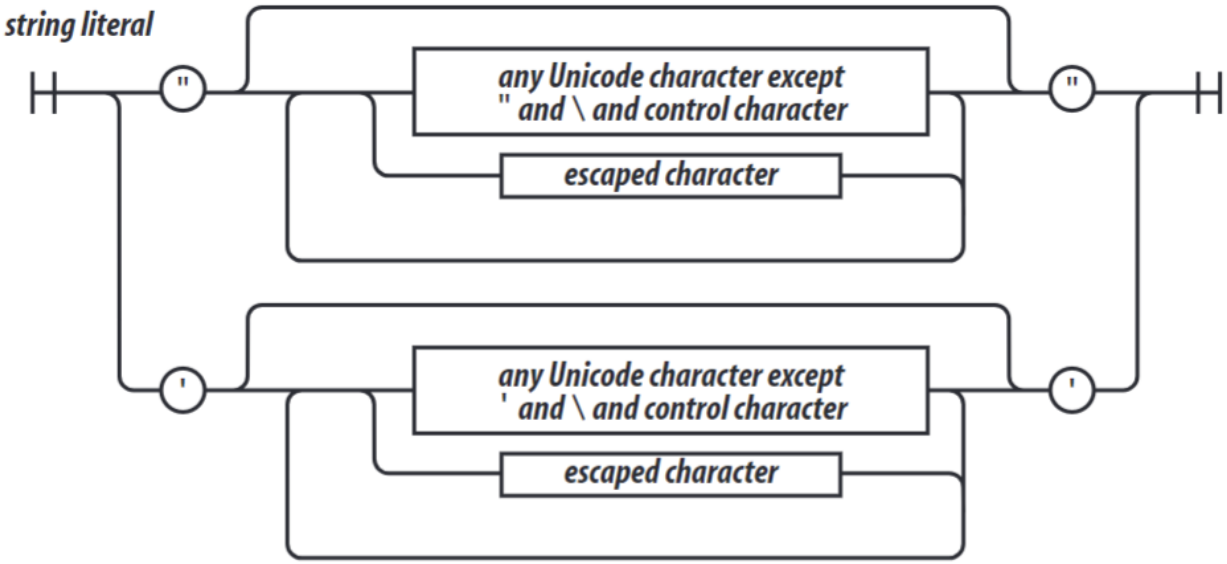
---

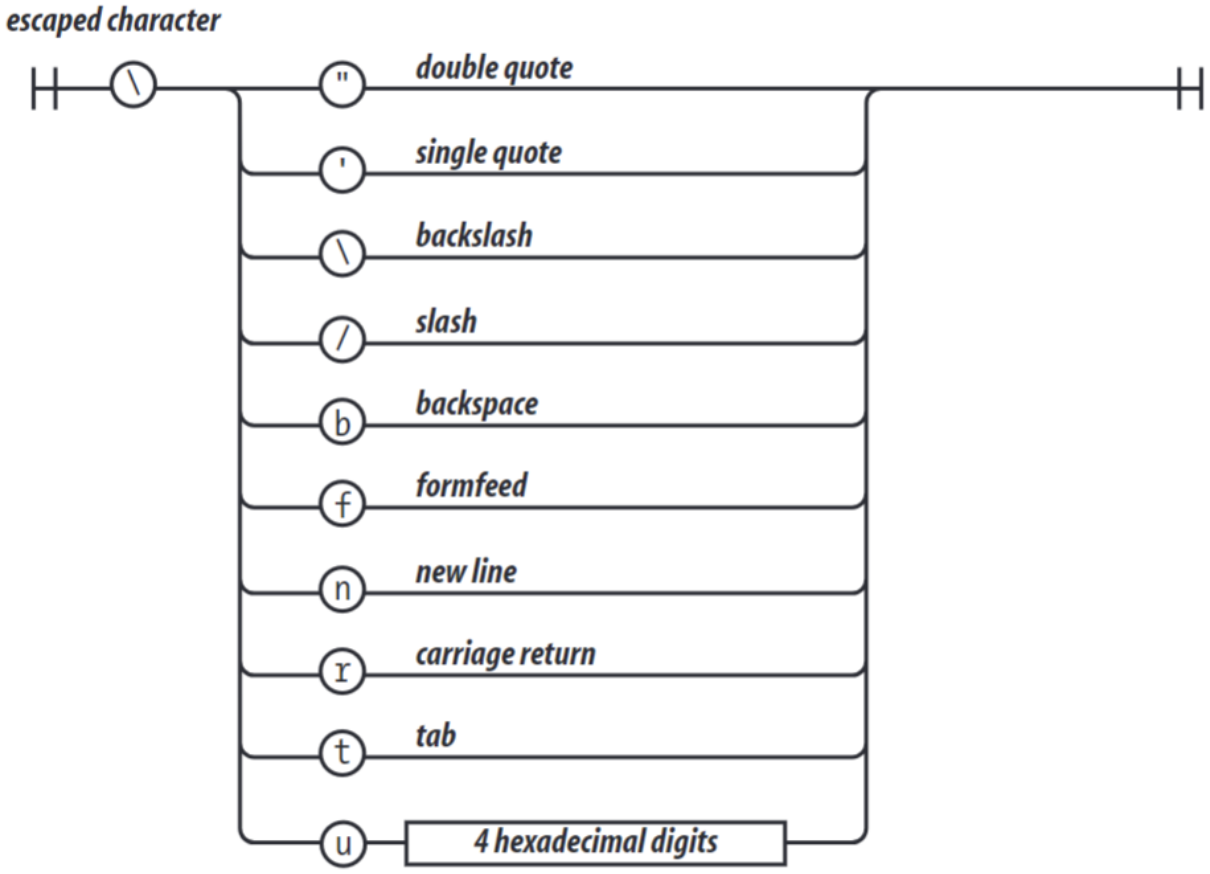
Strings

Strings are sequences of characters, the can be wrapped in single `''` or double quotes `""` and can contain zero or more characters
The `\` (backslash) is an escape character used to "escape" typical sequence. If you wanted to use single or double quotes in a string you would have to use the escape character

```
var myString = "She said \"Hi!\" to the boy."
```

JS does not have a character type, to do work on a character just use a string with one character in it.





All strings have a length property

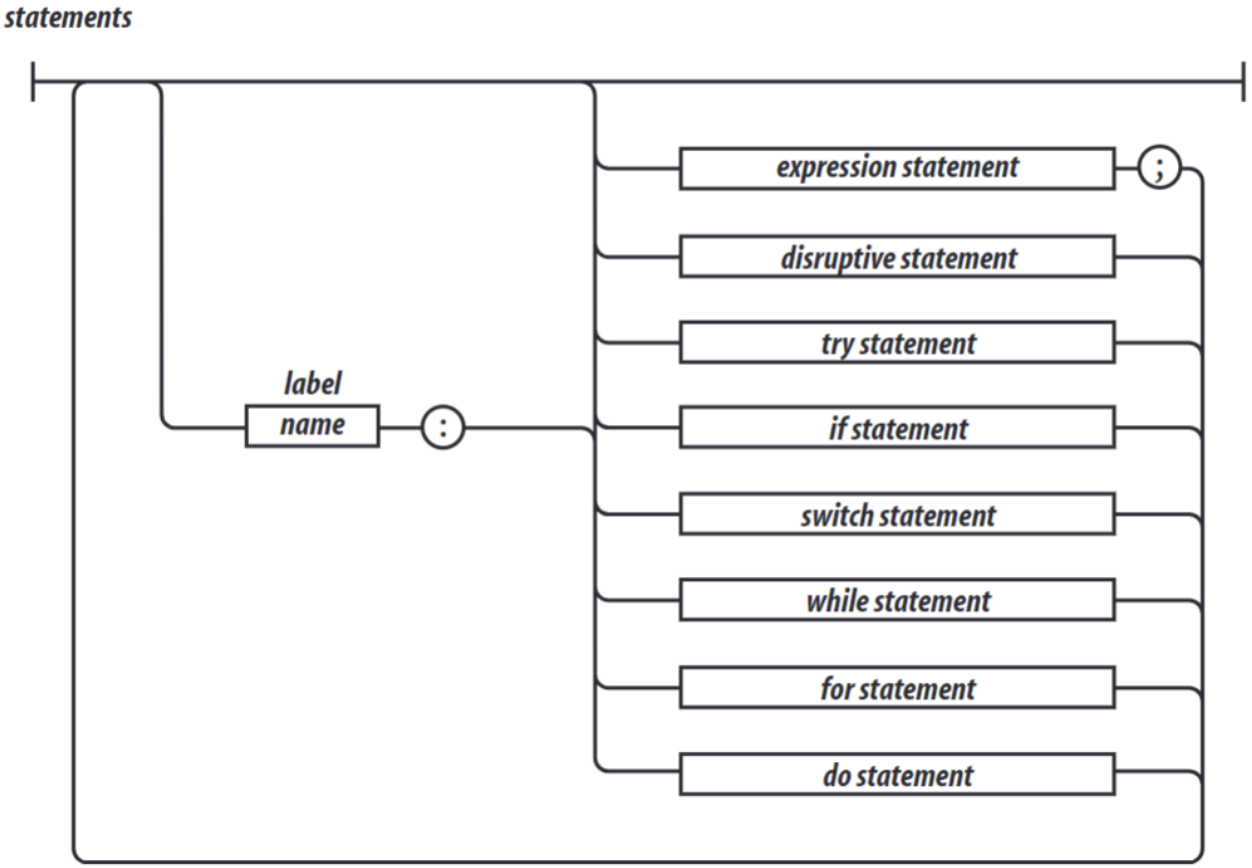
```
lengthOfString = "seven".length //this is equal to 5
```

Statements

Whenever JS is ran, each time the browser sees a `<script>` tag the JavaScript file linked to it is immediately compiled and executed.

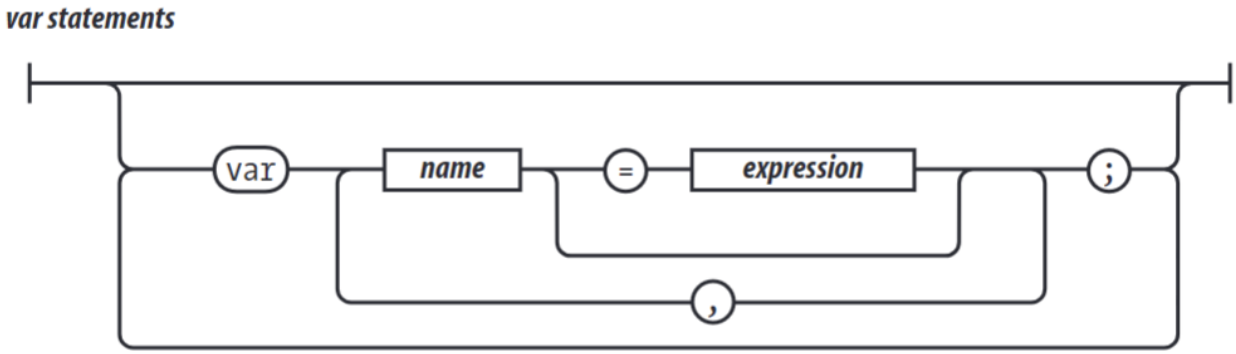
By default, statements are executed top to bottom, with exceptions:

> the use of `if` and `switch` statements to choose what to execute.
> the use of `for`, `while`, and `do` statements we can loop and and repeat statements.
> the use of `break`, `return`, and `throw` statements which interrupt execution.



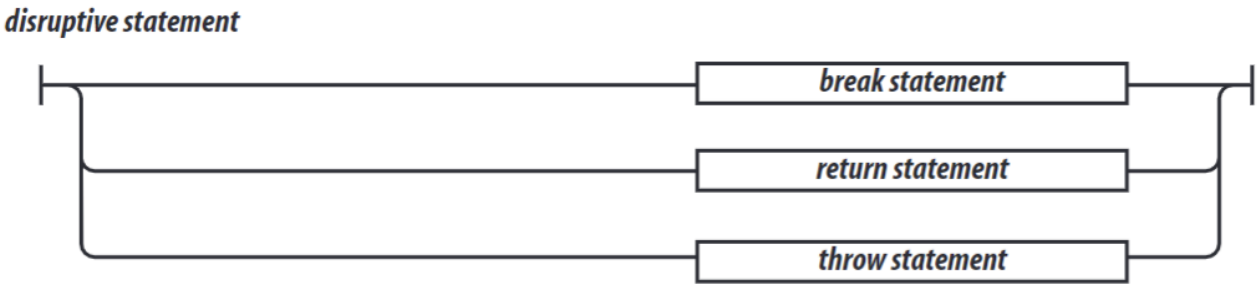
`var` Statements

> `var` statements are used for declaring variables



Disruptive Statements

Disruptive statements will break out of the current execution block and do various things

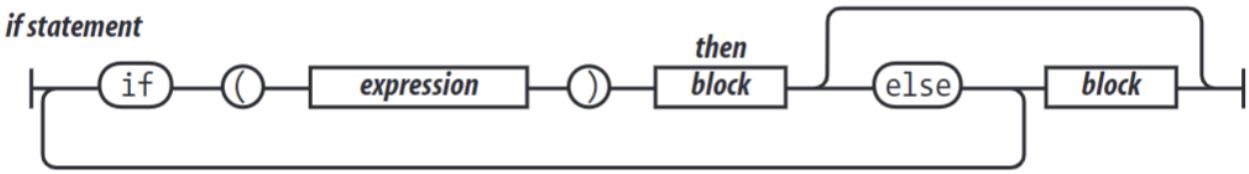


Blocks

Its important to note that blocks — created by `{}` do **not** define a new scope in JS. In other languages any variables declared within a scope will be deleted whenever you exit that scope, this is not the case for JS. Variables should be defined at the top of the function and not in blocks. The reason functions do define "scopes" is because nothing else can be executed while inside a function (unless asynchronously). So even though all variables within a function are global, they are essentially private since after the function terminates the only thing that makes it out of the objects deletion is what you export.

`if` Statements

`if` statements are used to alter the flow of execution based on conditions



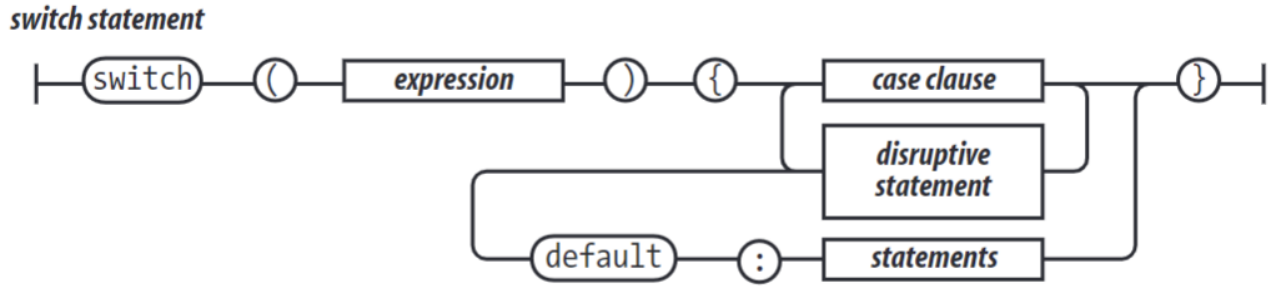
These are the *falsy* values:

- `false` • null
- `undefined` • `The empty string` "
- The number `0`
- The "number" `NaN`

All other values are *truthy* including `true`, the string `"false"`, and all objects

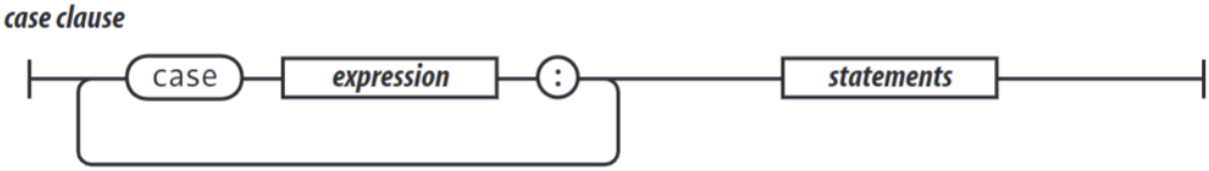
`switch` statement

`switch` statements are multi-branch if statements. Comparing conditionals and executing all that are true



`case` statement

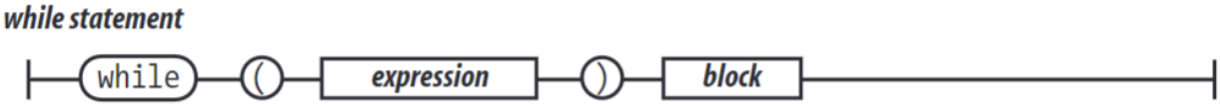
`case` is a subset of the switch statement which is what the expression in the switch is being compared to



Note that most of the time you will want to break in the case of a correct case being triggered as to note fall through and continue checking conditions, as well as accidentally running the `default` clause

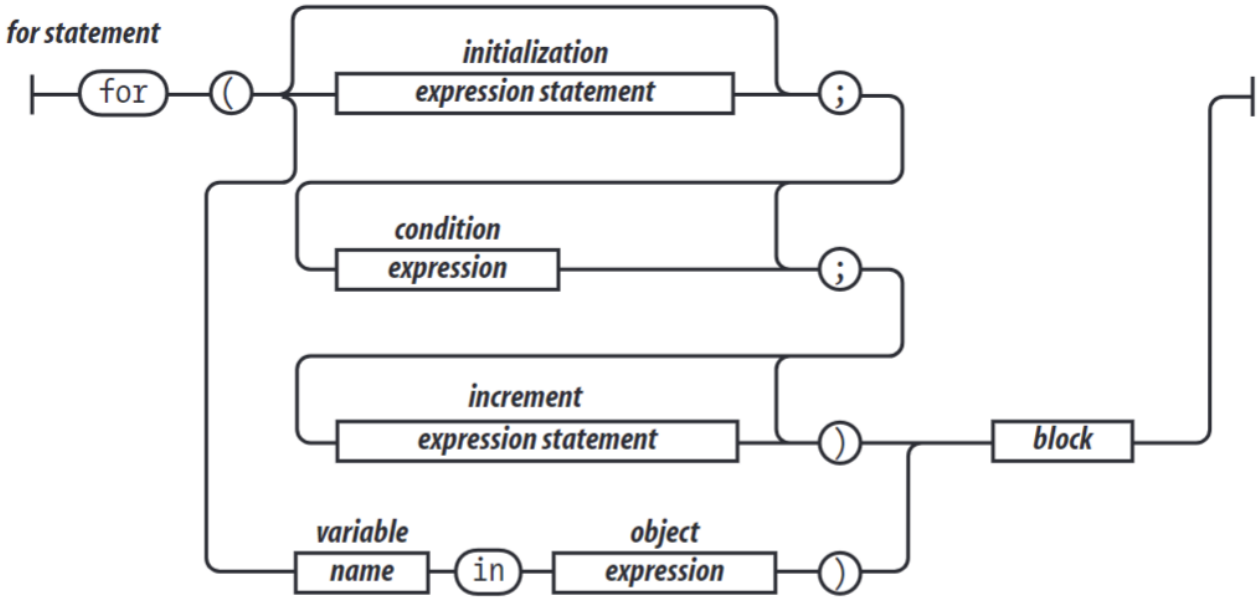
`while` statement

The `while` statement is the most simple of the loops, while the condition is truthy, it will continue execution of the block, whenever it is falsy, it will exit.



`for` statement

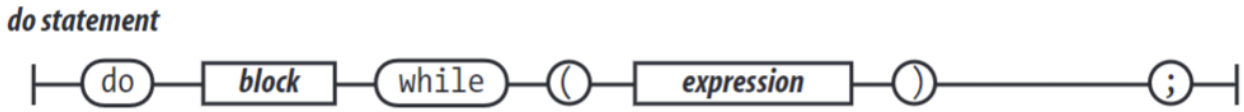
The `for` statement is a bit more complicated then the while statement. However, anything that can be done with a while statement can be done with a for statement and vice versa, its just a different flavor



The for loop also has another flavor which is short hand for iterating through all the keys of an object called `for in`

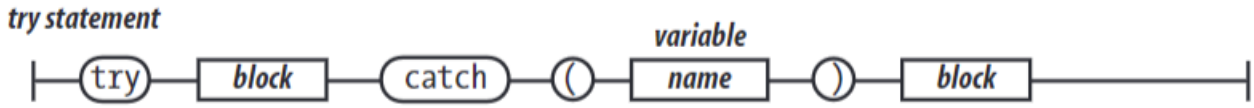
`do` statement

The `do` statement is a modified version of the `while` statement which executes the block before checking the looping condition



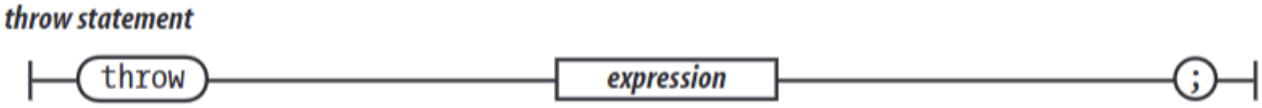
`try` statement

Executes the block of code and if it finds an error then it executes the second block



`throw` statement

The `throw` statement raises an exception. If in a try block, the catch condition is invoked.



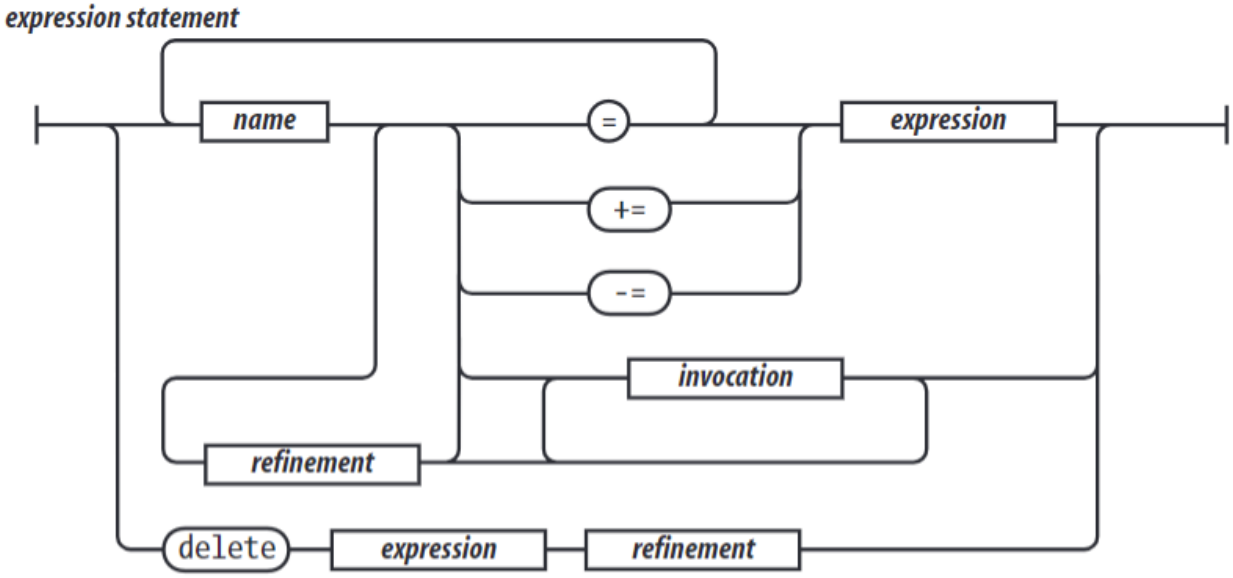
`return` statement

The return statement causes the given function to be exited and the value following to be returned

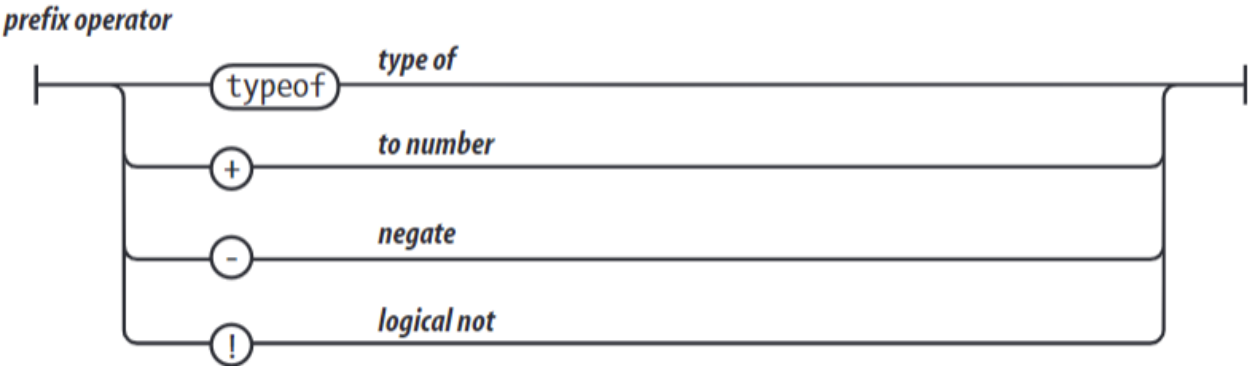
`break` statement

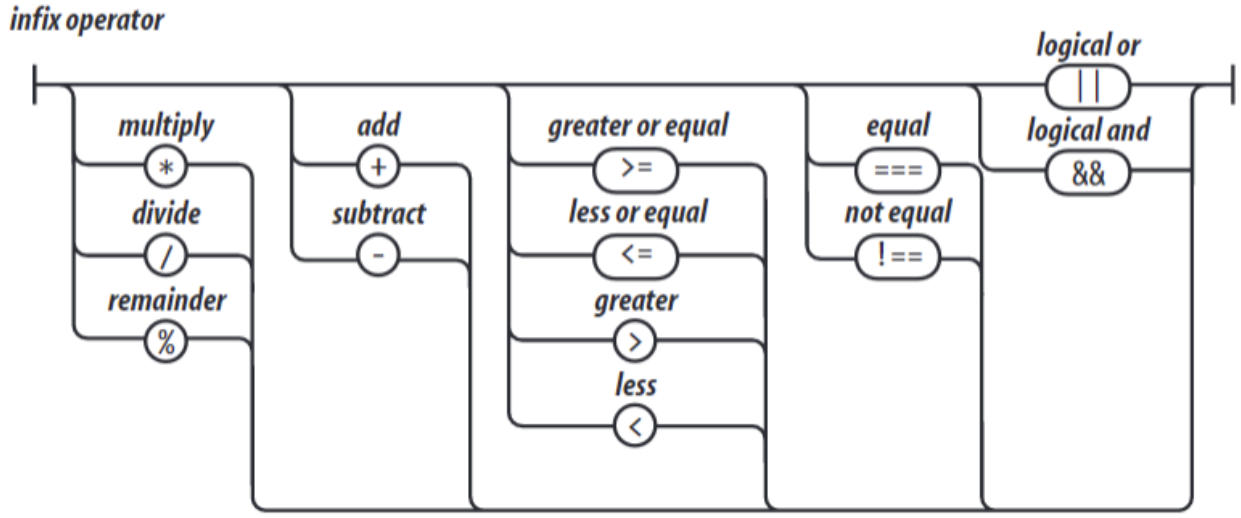
Break statement causes a forced exit from a loop or switch statement

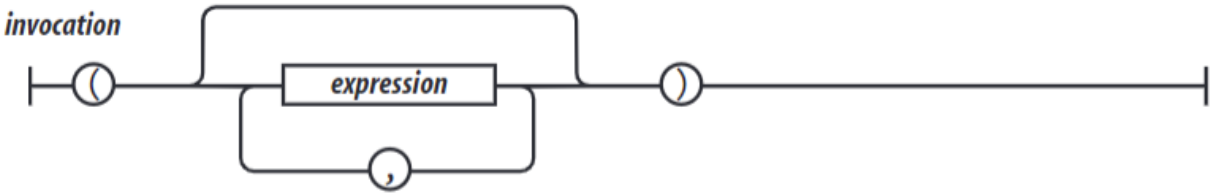
Expression statement

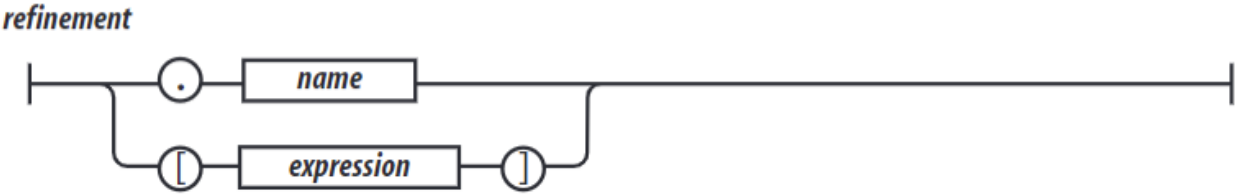


expression statements are used to either assign values to a variable, invoke a method, or delete a property from an object. Expressions are also often literal values — a string or a number.
Below are the prefix, infix, invocation, and refinement graphs



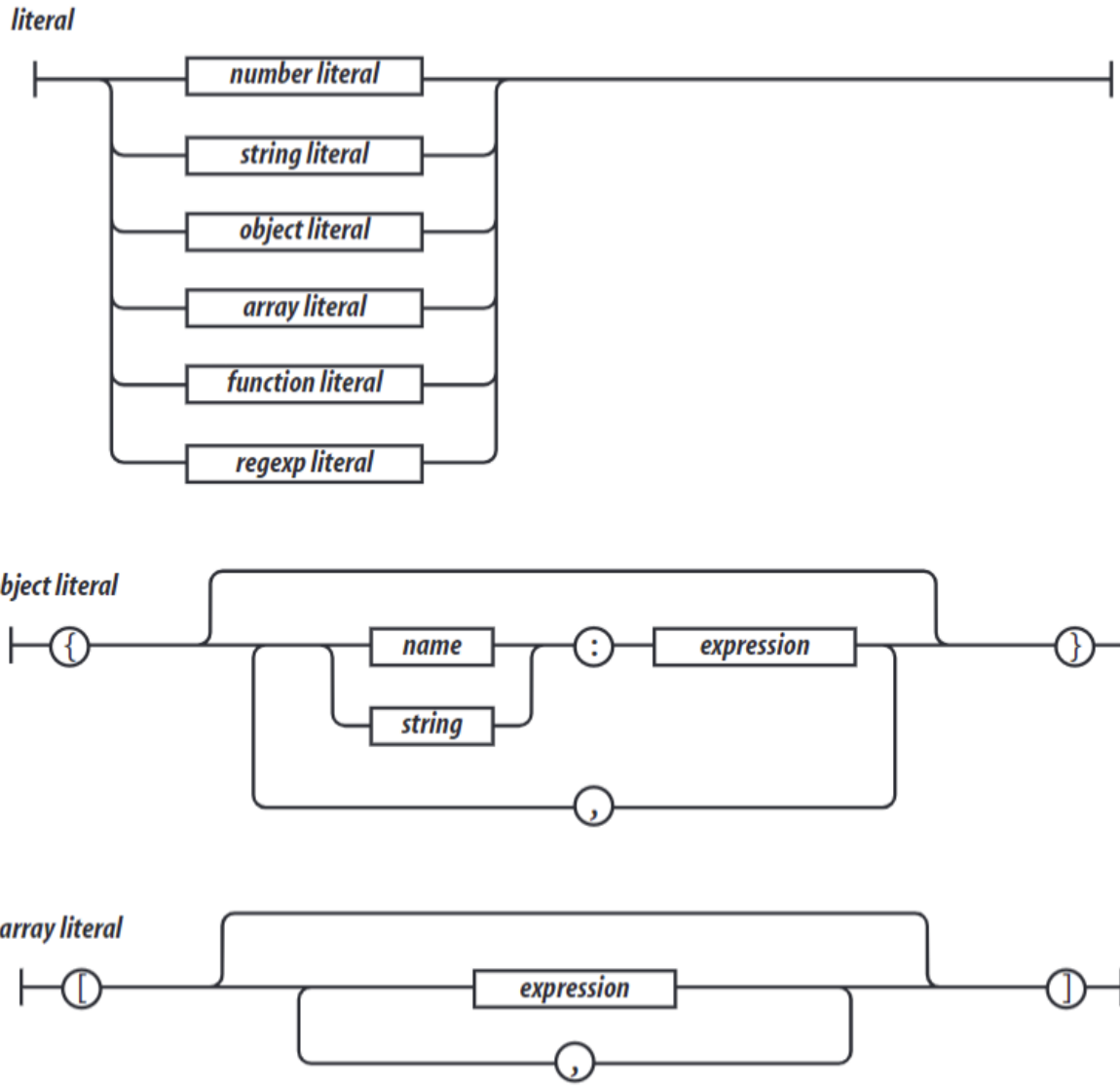






Literals

Literals are literal, things that actually are and not like a variable or anything of that nature. So you put a literal into a variable to store it


literal
number literal
string literal
object literal
array literal
function literal
regexp literal
object literal
{
name
:
expression
}
string
,
array literal
[
expression
]
,

Functions

Functions are objects, and functions are called upon for repeated actions