Crown

Current version: 2025-10-08

Crown is a metaprogramming syntax that can be implemented in any programming environment. You are currently viewing a document written in Crown for JavaScript.

The Crown engine runs code one command at a time, and maintains an 'implicit focus' at all times. This means that the results of the last command are always available for the next command, and so on. A sample Crown program might look like:

```
add 4 5, log The sum is [ current ] # prints 'The sum is 9' to the console
```

Getting Started

Download a copy of the Crown JavaScript Engine from http://localhost:3456/language/crown.js and add the following code to an HTML file:

```
<!doctype html>
<html>

<head>
 <meta charset="utf-8" />
 <title>My Website</title>
 <script type="text/javascript" src="./crown.js"></script>
</head>

<body>
 <noscript>JavaScript is required</noscript>
 <script type="text/javascript">
  async function main() {
   await crown().run`
    set document [ at document ]
    set greeting [ get document createElement, call h1 ]
    get document body appendChild, call [ get greeting ]
    set [ get greeting ] textContent 'Hello, world'
   `
   // or, if your Crown code is in a file:
   // await crown().runFile('./main.cr')
  }
  main().catch(e => console.error(e))
 </script>
</body>

</html>
```

If everything is set up properly, meaning the ./crown.js file is accessible, loading the HTML file in a browser will produce the following:

Hello, world

Crown Syntax

Crown has only 10 syntax characters: ⏎,[]()*'\#

Name	Symbol	Description
new line	`<line break>`	separates statements
comma	`,`	separates statements, is equivalent to a new line
left bracket	`[`	opens a block
right bracket	`]`	closes a block
left parenthesis	`(`	opens an expansion group
right parenthesis	`)`	closes an expansion group
asterisk	`*`	expands a list into separate items
single quote	`'`	opens or closes a string literal
escape	`\`	prevents a following ' from ending a string literal (use \\ for a literal \)
hash	`#`	marks the rest of the line as a comment

Crown Commands

There are many named commands in Crown, as follows:

Command	Example code	Description
#	`# a comment`	Creates a comment
<	`value 4, < 5`	Returns true if current is less than argument
<=	`value 5, <= 5`	Returns true if current is less than or equal to argument
=	`value 5, = 5`	Returns true if current is equal to argument
>	`value 6, >= 5`	Returns true if current is greater than argument
>=	`value 5, >= 5`	Returns true if current is greater than or equal to argument
add	`add 1 2 3`	Sums all arguments, and the focus if it is a number (example returns 6)
at	`at document body`	Sets focus to a property of focus (example will have focus set to <body> element)
call	`at alert, call Hello`	Invokes a function (example calls alert with 'Hello'). Updates the focus to the return value. To retain the function in focus and ignore the return value, use tell instead.
clone	`clone, set x 4`	Clones the current scope, retaining inheritance (in example, new scope has x set to 4)
comment	`log [ comment ]`	Returns the last comment (example returns 'a comment')
current	`value 4, log [ current ]`	Returns the implicit focus (example logs '4')
default	`get foo, default 4`	Returns the argument if the focus is undefined or null
divide	`value 4, divide 2`	Divides the focus by the argument(s)
do	`do [ call abc ]`	Run some code without modifying the focus

each	`each [ function item index [ ... ] ]`	Calls a function with two arguments for each item in an Array
entries	`entries [ function key value index [ ... ] ]`	Calls a function with three arguments for each key value pair in an Object
false	`value 4, > 5, false [ log No ]`	Runs the associated code block if focus is falsy This will log 'No', since 4 is not > 5
filter	`filter [ function item index [ ... ] ]`	Filters an array by calling a test function for each item in an Array
find	`find [ function item index [ ... ] ]`	Extracts a single item from an array by calling a test function for each item until a match is found
function	`function arg1 arg2 [ ... ]`	Creates a function with some named arguments (2 in example) and a function block
get	`log [ get foo ]`	Read the name from the current scope (example will log the value of foo)
global	`log [ global document title ]`	Read the name from the global scope (example will log the title of the current document)
group	`group [ function item [ get item key ] ]`	Groups an Array by the return value of the given function
is	`value 5, is 5`	The same as =

list	`list [ 1, 2, 3, 4 ]`
	`# or`
	`list 1 2 3 4`
		Creates an Array

load	`load ./path/to/file.cr`	Loads a Crown module into memory, but does not run it
log	`log foo is [ get foo ]`	Logs all arguments to the console (example logs 'foo is 4' assuming foo = 4)
multiply	`multiply 3 9`	Returns the product of all arguments, and the focus if it is a number (example returns 27)

not	`value 4, is 5, not`	Boolean invert of focus (example returns true)
object	`object [ a 1, b 2 ]`	Creates an object with given key-value pairs (example returns { a: 1, b: 2 })
point	`set a [ load ./file.cr, point ]`	Invokes focus with the scope as an argument (example runs file.cr, sets a to the result)
promise	`promise [ function resolve reject [ ... ] ]`	Returns a promise, which can be resolved or rejected later with e.g. [get resolve, call 4]
regexp	`regexp foo.*`	Creates a regular expression
run	`run 'log 42'`	Runs string argument as Crown code (example logs 42)
set	`set foo 4`	Sets a named variable in scope (example sets foo to 4)
subtract	`value 10, subtract 2 3`	Subtracts arguments from focus (example returns 5)
tell	`at alert, tell Hello`	Invokes a function (example calls alert with 'Hello'). Retains the function in focus and ignores the return value. To access the function's return value, use call instead.
template	`template 'Hello, %0' Human`	Inserts arguments into a string given as the first argument (example returns 'Hello, Human')
true	`value 6, > 5, true [ log Big ]`	Runs the associated code block if focus is truthy This will log Big since 6 > 5
unset	`unset foo`	Forgets a named variable in scope
value	`value 5`	Create a literal value (example returns 5)

About

Crown is developed by Nathanael S Ferrero. For help and support, or to report bugs so I can fix them, kindly send me an email at nate@tagme.in.