Noumenon

Manual

Tim Wiederhake

November 10, 2015

Noumenon is a dynamic, strongly typed script language. It is simple, yet powerful and extensible. In contrast to many other languages, there is no automatic type conversion in Noumenon.

Contents

[1] http://creativecommons.org/licenses/by/3.0/

1 Building Noumenon

Get a copy of Noumenon's source by either "git cloning":

```
$ git clone git@github.com:twied/noumenon.git
$ cd noumenon
```

... or downloading the sourcecode zip file:

```
$ wget https://github.com/twied/noumenon/archive/master.zip
$ unzip master.zip
$ cd noumenon-master
```

After that, you can build Noumenon:

```
$ make
```

If you do not have a make command at your disposal, simply compiling every cpp-file togehter should do the trick as well.

Noumenon is developed under Linux but *should* work on different platforms as well. If you experience any issues, please report[2].

2 Hello World

Start Noumenon from your favorite shell. You should see the following:

```
Noumenon 0.1
Copyright (C) 2015 Tim Wiederhake
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/
   licenses/gpl.html>
This is free software: you are free to change and
   redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Report bugs to <https://github.com/twied/noumenon/issues>.
```

Enter your first Noumenon program:

```
println("Hello world!");
```

[2] https://github.com/twied/noumenon/issues

You should see:

```
Hello world!
```

Exit by pressing Ctrl-C or Ctrl-D.

3 Data types

Noumenon knows eight different data types:

Array An array is a variable list of values. It is not necessary that all values have the same type. Even nested arrays are possible. The values in an array can change and the length of the array can change as well. Arrays are denoted by square brackets (`[]`) and values are separated by commas (`,`).

Bool A boolean value may either be `true` or `false`.

Float A float is an IEEE 754 floating-point number of double precision, your default `double` in C. To tell Noumenon that an entered number is a float, you have to include the dot, i.e. `1.0` is float, `1` is not.

Function Functions have a variable number of arguments and may or may not return a value. Missing arguments are filled up with `null` values. If the function is returning no value, an implicit `null` value is returned.

Int A signed 64 bit integral number, `signed long long int` in C.

Null The `null` value is generally an error value, expressing some kind of failure. It is safe to operate on `null`, even dereferencing it like an array or object as every operation on `null` results in `null` without any side effects.

Object An unordered mapping from IDs to values. As with arrays, the values do not have to be of the same type and nested objects are possible. The mappings of an object may be changed. Objects are denoted by curly brackets (`{}`), the ID is separated from the value by a colon (`:`) and multiple mappings are separated by commas (`,`).

String Dynamic string of arbitrary length. The usual escaping mechanisms work, `\n` will be replaced by a newline character, `\t` by a tab and so on.

4 Operators

For a full list of all permitted unary and binary operations, please see the output of `test_unop.nm` and `test_binop.nm` in the `examples` directory. Binary operators do not have a notion of "operator precedence", so concatenation of operations must be made unambiguous by inserting braces (`()`).

Unary minus (-) The unary minus is used to negate numerical values, i.e. `Int` and `Float` values. Trying to negate a value of a different type, especially `null` will result in `null`:

```
println(-1);
// -1

println(-(-1));
// 1

println(-7.0);
// -7

println(-null);
// null

println(-"text");
// null
```

Unary negation (!) The unary negation is used to negate boolean values. Trying to negate a value of a different type will result in `null`:

```
println(!true);
// false
```

```
println(!false);
// true

println(!(!true));
// true

println(!7);
// null
```

Addition (+) Addition will add two numerical values, concatenate strings and add values to arrays. Trying to add values of incompatible type will result in `null`:

```
println(4 + 7.5);
// 11.5

println(["one"] + "two");
// [one, two]

println("text" + 42);
// text42

println(42 + "text");
// null
```

Subtraction (-) Subtraction will subtract two numerical values or remove values from an array. Trying to subtract values of incompatible type will result in `null`:

```
println(7.0 - 11);
// -4

println(["one", "two", "one", "three", "one"] - "one");
// [two, three]

println(["two", "three"] - "one");
// [two, three]

println("hello" - "world");
// null
```

Multiplication (`*`), Division (`/`), Modulo (`%`) These operators work on numerical values (`Float` and `Int`), with the exception of Modulo, which works on `Int` values only. Division by zero returns `null` as well as any operation on incompatible types.

```
println(3 * 6.0);
// 18

println(17 / 3);
// 5

println(17 / 3.0);
// 5.66667

println(7 / 0);
// null

println(13 % 4);
// 1

println("hello" / "world");
// null
```

Logical And (`&&`), Logical Or (`||`) These operators work on values of type `Bool` and `Object`. On the latter, they perform like set theory's "cut" and "union" operators.

```
println(true && false);
// false

println(false || true);
// true

println({a: 1, b: 2} && {b: 3, c: 4});
// {a: 1, b: 2}

println({a: 1, b: 2} || {b: 3, c: 4});
// {a: 1, b: 2, c: 4}
```

Equality (`==`), Inequality (`!=`) Check if two values are of the same type and same content. Returns `null`, if the two values are not of the same type, even if the types are related, like `Int` and `Float`. Note that if at least one value is `null` the result is `null`.

```
println(1 == 2);
// false

println(2 != 3);
// true

println(1 == 1.0);
// null

println("two" == 2);
// null

println({a: 1, b: 2} == {a: 1, b: 2});
// true

println({a: 1, b: 2} == {a: 3, b: 2});
// false

println(null == null);
// null
```

Less (<), Less or equal (<=), Greater (>), Greater or equal (>=) Checks if the given inequality holds true. This only works on `Int` and `Float` values. Returns “null” if at least one value is of any other type.

```
println(1 < 4);
// true

println(2 >= 2.0);
// true
```

5 Built in variables and functions

print, println These functions will print out their arguments. `println` will also append a line break.

```
println("a", 1, 7.0, {foo: "bar"}, [2, 5.0, []]);
// a17{foo: bar}[2, 5, []]
```

typeof This function will return the type of its first argument as a `String`.

```
println(typeof(2));
// Int

println(typeof(2.0));
// Float

println(typeof("text"));
// String

println(typeof([]));
// Array
```

length This function will return the length of its first argument as an `Int`. This is the number of characters for `String`, the number of elements for `Array` and `Object` and `0` otherwise, even for `null`.

```
println(length("Hello world!"));
// 12

println(length([1, 2, 3, 4]));
// 4

println(length(7.0));
// 0
```

range This function will create an array containing all integers between the first argument's value (including) and the second argument's value (excluding). If called with less than two arguments or arguments that are not of type `Int` it returns `null`. If the first argument is greater than or equal to the second argument, an empty array is returned.

```
println(range(5, 10));
// [5, 6, 7, 8, 9]

println(range(10, 5));
// []

println(range(1));
// null
```

list This function is available in interactive mode only. It lists all defined variables in the current context.

arg This variable contains the program's argument as an `Array` of `Strings`.

env This variable contains an `Object` which holds the environment variables as they were when the program was started. Note that all elements are of type `String`.

require Include and execute a script. The first argument must be of type `String` and contain the file name of the script. Additional arguments are passed as the scripts `arg` variable. If the called script exits without a `return` statement, this function will return an empty `Object`, otherwise the value from the `return` statement. If the script was not found or failed, e. g. because of syntactical errors, the function returns `null`. For an example, see `require_main.nm` and `require_lib.nm` in the `examples` directory.

6 Language elements

Comments Comments are the same as in C or Java.

```
// this is a comment
/* this is
a comment */
```

var New variables are introduced with the `var` keyword. Variables need to have a value assigned instantly. This value and even the type of the value can be change later by assigning a new value to the variable. Variables are bound to a scope (each pair of curly brackets `{}`) and are available to inner scopes. Redefinition of a variable in the same scope is illegal, defining a variable with the same name of a variable of an outer scope shadows that variable.

```
var i = 7;
println(i);
// 7

/* var i = 2; ERROR */

if (3 < 4) {
    println(i);
    // 7

    var i = 4;
    println(i);
    // 4

    i = i + 1;
    println(i);
    // 5
}

println(i);
// 7
```

if, else Branching works as in every other language. Note that the curly brackets for the positive branch are mandatory unline e. g. C / Java. If the condition is not of type `Bool`, it is evaluated to `false`.

```
if (4 <= 3) {
    println("less or equal");
} else {
    println("not less");
}
// not less

var i = 7;
if (i < 5) {
    println("less than five!");
} else if (i < 10) {
    println("less than ten!");
} else {
    println("pretty big!");
}
// less than ten!

if ("Hello world!") {
    println(true);
} else {
    println(false);
}
// false
```

while The `while` loop executes the statements in the body as long as the condition holds true. The condition is evaluated at the beginning of each iteration.

```
var i = 5;

while(i > 0) {
    println(i);
    i = i - 1;
}
// 5
// 4
// 3
// 2
// 1
```

for The `for` loop is what would be a “for each” loop in other languages. It iterates over the elements of an array, the elements of an object or the characters of a string. With an alternate syntax, a second variable holds the index of the current element. Notice the use of the `range` function in the example.

```
for (var index, value : range(5, 10)) {
    println(index, " => ", value);
}
// 0 => 5
// 1 => 6
// 2 => 7
// 3 => 8
// 4 => 9

for (var c : "hello") {
    println(c);
}
// h
// e
// l
// l
// o

for (var index, value : {a: 1, b: 3.0, foo: "bar", array:
   []}) {
    println(index, " => ", value);
}
// a => 1
// array => []
// b => 3
// foo => bar
```

return Returns a value from a function (see section “Calling functions”). If `return` is called from the outer most scope (main program, so to speak), it will terminate the program. If the type of the returned value is `Int`, it will be used as exit code.

7 Calling functions

Functions are regular values in Noumenon. They can be assigned to variables like any other value. When a function is called, the argument values are assigned to the declared parameters. Excess values are discarded, missing parameters are assigned a value of `null`. There is no mechanism to create variadic functions, as an array could be used instead. If a function does not return a value, an implicit return value of `null` is substituted. Trying to “call” a value that is not of type `function` will return `null`.

```
var add = function(a, b) {
    return a + b;
};

println(add(7, 4));
// 11

println(add("hello ", "world"));
// hello world

println(add("text "));
// text null

println(add(1, 2, 3, 4));
// 3
```

Function arguments can be considered “call by value”, with the caveat that containers (`Arrays` and `Objects`) contain their elements as references and can therefor be changed. Adding or removing elements from the local container does not change the callers value.

```
var array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
var f1 = function(x) {
    x = x + 2;
}
var f2 = function(x) {
    x[0][0] = 0;
}

f1(array);
println(array);
// [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

f2(array);
println(array);
// [[0, 2, 3], [4, 5, 6], [7, 8, 9]]
```

8 Indexing variables

Variables of the types `Array`, `Object` and `String` can be indexed. If a variable of any other type is indexed, the resulting value is `null`. If the index is out of range, not of type `Int` (`Array` and `String`) or not of type `String` (`Object`) the result is `null` as well.

```
var s = "hello world";
var a = [7, 8, 9];
var o = {a: 1, b: 2, c: 3};
var i = 42;

println(s[4]);
// o

println(a[0]);
// 7

println(o["b"]);
// 2

println(i[0]);
// null

println(s[100]);
null

a[0] = 100;
println(a);
```

```
// [100, 8, 9]
```