Setanta syntax and semantics

The syntax of *Setanta* is new, but should feel familiar to most people. It has been designed to be simple and approachable. It takes inspiration from C like languages, but has some new ideas of it's own. The grammar has been designed for unambiguity so no semicolons or similar construct are required.

Setanta programs, like most imperative languages, consist of a sequence of statements. Some important *Setanta* features are outlined below:

A.1 Semantics

A.1.1 Paradigm

Setanta is classic imperatively executed language, it's multi-paradigm as it supports objects and inheritance, as well as higher order first class functions.

A.1.2 Type System

The *Setanta* type system is a strongly, but dynamically typed system. In practice this means the variables can hold any type of value, and the type of a variable can change over time, but that functions and operators will not try to automatically cast types to other types. This is in stark contrast to JavaScript which is famed for how weak it's typing system is, leading to lots of unintuitive errors.

The primitive types in *Setanta* include the familiar set of primitives: numbers (floating and integral), strings and booleans.

Setanta also supports functions and objects as first class entities.

A.1.3 Scoping & Closures

Setanta, like most modern programming languages, is lexically scoped and supports closures. By syntactically distinguishing between variable initialisation and variable assignment (Section A.2.1), the lexical scoping is easy to visually understand.

Listing A.1: Closures

```
1  gníomh adder(x) {
2      gníomh f(y) {
3          toradh x + y
4      }
5      toradh f
6  }
7
8  addfive := adder(5)
9  scríobh(addfive(2))
10 >-- Outputs 7
```

A.1.4 Inheritance

Setanta uses a traditional single dispatch, class based single inheritance system. In practice this means that a class `A` can inherit from a single class `B`. `A` inherits all of `B`'s methods, including constructor. If

`A` redefines any of `B`s methods, including the constructor, it can access the methods of `B` using the tuis keyword.

A.2 Syntax

A.2.1 Variable declarations

In *Setanta* variables are declared using the `:=` operator, and can be re-assigned using the classic `=` operator. The distinction is to provide a clear lexical difference between variable declaration and reassignment.

Listing A.2: Variable declaration and assignment

```
1 x := 0
2 x = x + 1
```

A.2.2 Literals

Setanta supports literals for integers, booleans, null, strings, lists and null.

Listing A.3: Setanta literals

```
1 a := 500
2 b := 'Dia duit domhan'
3 c := [1,2,3,4, fíor ]
4 d := fíor != breag
5 c := neamhní
```

A.2.3 Expressions

Setanta expressions are built from literals, identifiers, and a large selection of operators.

The familiar arithmetic operators are included, `+,-,*,/,//,%` meaning addition, subtraction, multiplication, division, integer division and modulus respectively. The operators `==, !=, <, >, <=, >=` can be used for comparisons. `(` and `)` can be used for grouping.

Functions are called by using appending a bracket enclosed sequence of arguments to the function identifier e.g. `function(arg1, arg2)`.

Lists and strings can be indexed with the traditional square bracket notation, `arr[index]`.

Object access is done with the `@` operator. If some object `obj` has a member or method `a`, this is expressed as `a@obj`. Note that this is inverse from the familiar dot `(.)` operator.

Listing A.4: Setanta expressions

```
1 2 + 3 - (5 * 3)
2 4 >= 2
3 4 % 2 == 0
4 scríobh('hey')
5 array[1 + 2][0] + 'hey'
6 a@b@c@obj != 4
```

A.2.4 Assignment

As well as variables, lists and objects can be assigned to in the way you would expect

Listing A.5: Valid Setanta assignments

```
field@obj = 'dia duit'
arr[0][0] = 4 + 5
field@obj(arg1)[1] = 42
```

A.2.5 Conditionals

Setanta support the classic conditional execution structure of `if` → `else-if` → `else`. This is mostly a direct translation into Irish, as it uses the keyword **má** meaning "if", and the word nó meaning "or" However it should be noted that no bracketing is required around the expression.

Listing A.6: Setanta conditionals

```
má x == 0
    scríobh('Tá x cothrom le 0')
nó má x == 1
    scríobh('Tá x cothrom le 1')
nó
    scríobh('Tá x níos mo ná 1')
```

A.2.6 Loops

Setanta supports two main types of loops, "le idir" loops that allow the user to specify start and ends to the loop, and "nuair-a" loops, which are the familiar while loops.

"le idir" take the form **le** i **idir** (a, b, c), which translates as roughly "with i between (a, b, c)". This is similar to Python's **for** i **in range**(a,b,c) loop, meaning the loop starts loops from `a` to `b`, with step size `c`. The current iteration would be available in the variable `i` inside the loop.

"nuair-a" translates roughly as "when". This loop is equivalent to a `while` loop from many C-family languages.

Listing A.7: Setanta loops

```
i := 0
le i idir (0, 10)
    i = i + 1
x := 0
 nuair-a  x < 10
    x = x + 1
```

A.2.7 Functions

Functions in *Setanta* are referred to with the term "gníomh" meaning "action". They can constructed with the **gníomh** keyword, followed by a name and argument list. The toradh keyword can be used to return values from the function.

Listing A.8: Setanta function example

```
gníomh fibonacci(n) {
```

```
2     má n <= 1
3         toradh 1
4     toradh fibonacci(n-1) + fibonacci(n-2)
5 }
```

A.2.8 Classes

Setanta supports declaring new classes, with methods, and a constructor. Classes can inherit from other classes using the keyword *ó*. Methods are declared as functions within the body of the class. A method with the name `nua` defines the constructor, "nua" translates as "new" in English. The keyword `seo` can be used to access member fields and methods of the class, "seo" is a direct translation of the familiar "this" keyword.

Listing A.9: Setanta classe example

```
1 creatlach Person ó Animal {
2     gníomh nua(name) {
3         name@seo = name
4     }
5     gníomh speak() {
6         scríobh('Hi, My name is ' + name@seo)
7     }
8 }
```