Anubis programming language

# Comments:

// One line comment...

/\* Multi-line comment... \*/

# Identifiers:

* Valid characters:
  + Letters: A-Z Ñ Á É Í Ó Ú a-z ñ á é í ó ú
  + Digits: 0-9
  + Underscore: \_
* Must start with letter or underscore.

# Basic types:

* Boolean: **Bool**
* Character: **Char**
* Number: **Integer**, **Real**
* String: **String**, [**Char**]
* Function: **Function**
* Tuple: **Tuple**, <>
* List: **List**, []
* Object: **Object**, {}

# Literal expressions:

* Bool: true, false
* Integer: 0, 1, 2, ...
* Real: 1.2e3 -> #.#[{e|E}[+|-]#]
* Char: 'a', 'b', ...
* String: "hello"
* Hexadecimal: 0x09AF -> 0{x|X}# (# = 0..9, {a..f|A..F})
* Null: nil, null
* Tuple: <value, ...>
* List: [value, ...]  
   [initial-value..final-value]
* Object: { {name|"name"} : value, ... }

Special characters:

* \0 Null character
* \\ Backslash
* \' Single quote
* \" Double quote
* \n New line
* \t Tab
* \r Carriage return
* \x ASCII character in hexadecimal (\xFF).
* \u Unicode character in hexadecimal (\uFFFF).

# Operators:

|  |  |  |  |
| --- | --- | --- | --- |
| **Precedence** | **Operator** | **Description** | **Associativity** |
| 0 | () | Parenthesis | none |
| 2 | ++  --  ()  []  ..  .  as  is  in | Postfix increment  Postfix decrement  Function call  Array access  Subrange  Member access  Conversion  Type check Check existence | left-to-right |
| 4 | ++  --  +  -  !  not  ~ | Prefix increment  Prefix decrement  Unary plus  Unary minus  Logical NOT Logical NOT  Bitwise NOT | right-to-left |
| 6 | \*  /  % | Multiplication  Division  Remainder | left-to-right |
| 8 | +  - $ | Addition  Subtraction List concatenation |
| 10 | :: | List front insert |
| 12 | <<  >> | Bitwise left shift  Bitwise right shift |
| 14 | <  <=  >  >= | Relational <  Relational ≤  Relational >  Relational ≥ |
| 16 | ==  !=  <> | Relational =  Relational ≠  Relational ≠ |
| 18 | & | Bitwise AND |
| 20 | ^ | Bitwise XOR |
| 22 | | | Bitwise OR |
| 24 | &&  and | Logical AND |
| 26 | xor | Logical XOR |
| 28 | ||  or | Logical OR |
| 30 | ?: | Ternary conditional | right-to-left |
| 32 | =  +=  -=  \*=  /=  %=  <<=  >>=  &=  ^=  |= | Simple assignment  Assignment by sum  Assignment by difference  Assignment by product  Assignment by quotient  Assignment by remainder  Assignment by bitwise left shift  Assignment by bitwise right shift  Assignment by bitwise AND  Assignment by bitwise XOR  Assignment by bitwise OR |
| 38 | , | Comma | left-to-right |

# Special operators:

* Member access: name.member
* Array access: [start[:end[:jump]]]  
   (NOTE: If the end value is given a sublist will  
   be returned with the elements from the start  
   to (end – 1) position.)
* List construction: item :: [list]
* List addition: [list] $ [list]
* New object: name = **new** type[([parameters])];
* Conversion: expression **as** type
* Type check: expression **is** type

# Declarations:

[**const**] name [-> type] [= value], name2 [-> type] [= value], ...;

[**var**] name [-> type] [= value], name2 [-> type] [= value], ...;

**alias** type[[]] name;

# Control structures:

Conditional:

**if** (condition) ...; [**else** ...;]

**switch** (variable) {

**case** value: ...; **break**;

[...]

[**default**: ...;]

}

Loops:

**while** (condition) ...;

**do** { ...; } **while** (condition);

**repeat** { ...; } **until** (condition);

**for** (variables; condition; expression) ...;

**for** (variable : list)...;

Errors:

**try** { ...; } **catch** ({...|variable}) { ...; } **finally** { ...; }

**throw** value;

Jump:

**break**;

**continue**;

Other:

**include** "file.asc";

**yield** expression;  
**return** expression;

List comprehension:

[expression <| list of assigns & conditions <; ...]

[x <| x <- [1,3..10]]

[x <| x <- [1,2,4,8,16]]

[(x,y) <| x <- [1,3] <; y <- [2,5,8]]

[x^2 <| x <- [1..10] <; even x]

# Functions:

Functions with name:

{ **function** | **func** } name (parameters) [: type] {

...; [**return** expression;] ...;

}

Anonymous functions:

(parameters) => { ...; [**return** expression;] ...; }

Parameters:

* Parameters: [**ref**] match-expression 1, ... [, args...]
* Match expression:
  + Anything: name
  + Typed parameter: name -> type
  + List of anything: x :: xs
  + Typed list: x -> type :: xs -> type  
     x -> type :: xs  
     x :: xs -> type
  + Tuple: ( name [-> type], ... )
* The variable arguments must be at the end.

# Classes 1.0:

Definition:

**class** name [(fathers)] {

declarations...

}

Declaration definition:

[**[-]**|**[+]**|**[**[**-**|**+**]**@]**] declaration

* By default everything is public.
* - == private.
* + == protected.
* @ == static.
* Constants are treated as static members.

Constructor & destructor:

{ **constructor** | **ctor** } (parameters) { ...; } // Called on new.

{ **destructor** | **dtor** } (parameters) { ...; } // Called when removed.

Special operators:

**self**

**base**[[father]]

Properties:

{ **property** | **prop** } name [: type] {

{ **get**; [**set**;] | **get** { **return** variable; } [**set** { variable = **value**; }] }

}

# Classes 2.0:

Properties:

{ **property** | **prop** } **self** [index] [: type] {

{ **get**; [**set**;] | **get** { **return** variable; } [**set** { variable = **value**; }] }

}

Operators overloading:

{ **operator** | **oper** } operator (parameters) [: type] {

...; **return** expression; ...;

}