Sugar Language Quick Reference

version 0.8 (jan-08) - Sebastien Pierre

Primitive types

Numbers	`1  1.2  -1.3 .123 0.123 0xFF96`
Strings	`"hello" 'hello' "\n\r\t"`
Lists	`[] [1] [1,2] ["a", ["b", 1]]`
Maps	`{} {one:1, two:2, three:3}`

Note: in lists and maps, the comma can be replaced by a newline, if the content is indented

Symbolic values

Undefined value	`Undefined`
Absence of value	`None`
Booleans	`True, False`
Not a number	`NaN`
Operation status	`Error, Success, Timeout`
Wildcard	`_`

Closures

Basic	`{a,b,c|print (a,b,c)}`

Multiline	`{a,b,c|`
	`  var d = a + b + c`
	`  print (d)`
	`}`

Empty	`{|}`

Basic operations

Allocation	`var a   var a:Number   var a = 1`
Computation	`a + 1   not v`

Invocation	`f ()   f (1, 2)   f (1, b=2, c=3)`
	`f (...[1,2,3])`
	`f (...={a=1,b=2,c=3})`

Instanciation	`new Rectangle (320,200)`
Resolution	`r width`
Slicing	`a[0]  a[:]  a[1:-1]  a[:-1]  a[1:]`
Iteration	`[1,2,3] :: {val,ind|print (val, ind)}`
Enumeration	`1..10   (a)..(a+20)   (-a)..(a)`

Computations

Basic algebra	`1+1 1-1 1/2 1*3 1^3  1%3`
Basic comparison	`1<2  2>3 1==1 1!=2`
Logical combinators	`1 and 1 or 3 and not 0`
Value identification	`a is b`
Value unification	`a like b`
Type identification	`a isa Number`
Slot identification	`a has length`

Control structures

Conditionals	`if EXPRESSION -> STATEMENT` `if EXPRESSION \n BLOCK \n end`

Invocation	`if EXPRESSION \n if ... \n else \n ... \n end`
	`for v in 0..10 \n BLOCK \n end`
	`for v, i in [1,2,3] \n BLOCK \n end`

Instanciation	`for v, k {a=1,b=2,c=3}  \n BLOCK \n end`
Resolution	`while EXPRESSSION  \n BLOCK \n end`

Control flow operations

Termination	`return EXPRESSION`
Generation	`yield EXPRESSION`
Interruption	`break  continue`

Exceptions

Throwing	`raise EXPRESSION`
Catching	`try \n BLOCK \n catch TYPE \n BLOCK \n end`

Style Guide

1. Identation matters (as in Python)
2. Documentation will make your code better
3. Classes as `CamelCase`
4. Functions, Methods and Invocations as `mixedCase`
5. Modules as `lowercase`
6. Shared properties and constants `UPPER_CASE`
7. Local variables `lower_case`
8. Put a space before parens

Idioms

Optional parens	`f (1) == f 1  f ("Hello")  f "hello"`
Cool one-liners	`0..10 :: {v,i|print (v,i)}  {|f();g();h()}()`
Iterate on anything	`[1,2,3] :: {v,i|}  {a=1, b=2} :: {v,k|}  r :: {v,i|}`
Design by contract	`@pre, @post, @always EXPRESSION in functions`
Invocation guards	`@when EXPRESSION`

Keywords

Variable declaration	`var`	Iterations/Repetitions	`for while`
Operators	`as and or not has is in`	Control flow	`return break continue yield`
Instanciation	`new`	Exceptions	`raise try catch finally`
Conditionals	`if else`	Terminator	`end`

Module declaration

```
Fully qualified name  @module org.sugarlang.core
Annotations           @author Sebastien Pierre
                      @version 1.0
Dcoumentation         | Documentation

Imports               @import org.sugarlang.datatypes
                      @import org.projecta.A as PrA
                      @from org.myproject import A, B, C

Globals               @shared DATA
Classes               @class...
Functions             @function...

Initialization        @init
                        DATA = new ...
                      @end

Main function         @main
                          print "Hello !"
                      @end
```

Function declaration

```
Name and args         @function f:Number a, b=1, c=3
Annotations           @pre   a<b+c
Documentation         | Documentation
  Comment                 # Comment
  Statements              var d = 1
                          ...
  Termination             return d * b + c
Explicit end          @end
```

Class declaration

```
Name and parent       @class Rectangle:Shape
Documentation         | Documentation
  Shared property         @shared    COUNT=0
  Instance property       @attribute w:Number
                          @attribtue h:Number
  Constructor             @constructor w,h,x=0,y=0
    Super invocation          super (x=x, y=y)
    Explicit self ref         self w = w ; self h = h
    Implicit ref              COUNT += 1
                          @end
  Instance method         @method getArea
                              return w * h
                          @end
  Class method            @operation getCount
                              return COUNT
                          @end
Explicit end          @end
```