

目 录

算术与逻辑操作符.....	2
预定义算术函数.....	3
位操作	3
条件控制函数.....	4
循环控制函数.....	5
选择函数.....	6
循环结构中的局部变量.....	6
返回函数.....	6
正则表达式.....	6
特殊字符，注释，空格，圆括号和其它标记法的使用	8
文件系统接口.....	10
端口	11
I/O	12
文件输入输出示例.....	13
数字	14
字符	16
字符串	16
逻辑值	18
数组 array.....	18
列表 list.....	18
关联表格 table	20
Cadence 数据库数据 dbid.....	20
定义函数.....	20
局部和全局变量.....	21
返回值	22
输入参数定义.....	22
函数的限制.....	24
示例	24

算术与逻辑操作符

函数名	语法	对应操作符
Data Access		
arrayref	a[index]	[]
setarray	a[index] = expr	
bitfieldl	x<bit>	<>
setqbitfieldl	x<bit>=expr	
setqbitfield	x<msb:lsb>=expr	
quote	'expr	,
getqq	g.s	.
getq	g->s	->
putpropqq	g.s=expr, g->s=expr	
putpropq	d~>s, d~>s =expr ~>	
Unary		
preincrement	++s	++
postincrement	s++	++
predecrement	--s	--
postdecrement	s--	--
minus	-n	-
not	!expr	!
bnot	~x	~
Binary		
expt	n1 ** n2	**
times	n1 * n2	*
quotient	n1 / n2	/
plus	n1 + n2	+
difference	n1 - n2	-
leftshift	x1 << x2	<<
rightshift	x1 >> x2	>>
lessp	n1<n2	<
greaterp	n1>n2	>
leqp	n1<=n2	<=
geqp	n1>=n2	>=
equal	g1 == g2	==
nequal	g1 != g2	!=
band	x1 & x2	&
bband	x1 ~& x2	~&
bxor	x1 ^ x2	^
bxnor	x1 ^^ x2	^^
bor	x1 x2	
bnor	x1 ~ x2	~

and	x1 && x2	&&
or	x1 x2	
range	g1 : g2	:
setq	s = expr	=

对于既有函数名又有操作符函数，采用函数名调用和采用操作符调用的效果是一样的。

示例 lessp(3 1)=>nil 等同于 3<1=>nil

预定义算术函数

语法	结果
通用函数	
add1(n)	$n + 1$
sub1(n)	$n - 1$
abs(n)	绝对值
exp(n)	自然对数的 n 次方
log(n)	
max(n1 n2 ...)	
min(n1 n2 ...)	
mod(x1 x2)	取模运算
round(n)	
sqrt(n)	
sxtd(x w)	将 x 右扩展 w 个符号位
zxtd(x w)	和上面的功能相同，速度更快
三角函数	
sin(n), cos(n), tan(n)	
asin(n), acos(n), atan(n)	
随机数生成函数	
random(x)	产生 $0 \sim x-1$ 的随机数
srandom(x)	初始化随机数生成函数的 x

位操作

含义	操作符
位与	&
位或	
位异或	^
左移位	>>
右移位	<<
取反	~

条件控制函数

if, when, unless, cond

- if

if 函数的语法结构

```
if( condition1 then      ; 当 condition1 为真值时执行代码块 expA
    expA
else
    if( condition2 then  ; 当 condition2 为真值时执行代码块 expB
        expB
    else
        ...
        expC
    )
); end if
```

- when

when 函数语法结构

```
when( condition          ; 当 condition 为真值时执行代码块 expA,
否则跳过
    expA
); end when
```

- unless

unless 语法结构

```
unless( condition        ; 当 condition 为假值时执行代码块 expA,
否则跳过
    expA
); end unless
```

- cond

cond 语法结构

```
cond (
    ( condition1 expA... ) ; 当 condition1 为真值时执行代
码块 expA
```

```
        ( condition2 expB ... ) ; ...
        ( condition3 expC ... ) ; ...
        ( t expN )                ; 之前的条件都不满足时默认执
行 expN
    ); end cond
```

[注]: “;” 表示注

循环控制函数

while, for, foreach

- while

while 语法结构

```
while( condition is true ; 当 condition 为真值时反复执行代码块
expA, 否则结束循环
    expA
)
```

- for

for 语法结构

```
for( i 1 5
    println(i)
)
```

将输出

```
1
2
3
4
5
```

- foreach

foreach 是简化版的 for，上面的例子用 foreach 写如下

```
alist = '(1 2 3 4 5)
foreach( item alist
    println( item )
```

)

foreach 通常用于操作列表，因为它会遍历列表的每一个元素。

选择函数

- caseq 的用法类似 cond，功能也类似。

```
caseq( var
      (condition1 expA)
      (condition2 expB)
      ...
    ) ; case 语法结构
```

循环结构中的局部变量

使用关键字 prog 可以在循环结构中定义局部变量。关于 prog 的详细介绍请参考函数章节。

返回函数

表示返回值的关键字是 return。return 可以强制一个 prog 结构立即结束，送出返回值，忽略剩余的循环操作。

```
示例 prog( ( )
            for( i 0 10
                when( oddp( i )
                    return( i )
                ) ; when
            ) ; for
        ) ; prog
```

这个 for 循环不会运行 10 次，首次 oddp(i) 为真值的时候循环就会结束。

正则表达式

在很多的 skill 应用中都会用到正则表达式，比如在一个字符串数组中查找带有关键字 user 的字符串。示例：

```
lStr = '("hello" "whenuser" "usercome" "ffus er" "ppuserd" "uster");
lRes = nil
foreach( item lStr
  if( rexMatchp("user" item) then
    lRes = cons( item lRes)
    println(item)
  )
)
```

输出:

```
"whenuser" "usercome" "ppuserd"
lRes =>("ppuserd" "usercome" "whenuser").
```

这里的关键命令是 rexMatchp, 函数形式为 `rexMatchp(t_pattern S_target)`, 这个命令将在第二个参数中查找满足第一个参数(模式)的条件, 如果找到返回真值, 否则返回假值。

- 正则表达式(pattern)的组成字符

语法	含义
c	匹配任意非特殊字符
.	匹配任意字符
\	用在特殊字符前面代表特殊字符本身, 用在 <, >, (,), and
1,...,9	的含义在下面描述
[c...]	一对方括号中包含一些字符将匹配括号中的任意一个字符, 如果第一个字符是 ^ 则表示不匹配括号中的所有字符, [a-z]表示匹配 a 到 z 之间的任一字符, "-" 表示范围。
*	用在一个以上描述的字符或 pattern 后表示匹配该字符或 pattern 0 次或多次。
+	类似*, 区别在于匹配至少一次
\(.\.)	如果包含在这个符号中的字符串匹配上了, 被匹配的字符可以被提取出来(最多支持 9 中匹配)
\n	和上一个符号对应, 反斜杠后跟一个数字将可以提前到上一个符合保留的字符串
\<...\>	这个符号表示完全匹配一个字符串的开始和结尾
rs	2 个正则表达式的组合将尽可能多的匹配 r 并匹配一个 s
^, \$	^表示字符串的开始, \$表示字符串的结尾

- 常用正则匹配函数

rexCompile, rexExecute, rexMatchp, rexSubstitute 示例: 使用 rexCompile 和 rexExecute 实现开头的示例

```
lStr = '("hello" "whenuser" "usercome" "ffus er" "ppuserd" "uster");
```

)

定格式的列表中提取 symbol 名字和对应的坐标

) ;

将输出：

R754<->2770.00<->-2730.00

法的使用

- 特殊字符

SKill 中的特殊字符列表

字符	全名	含义
\	反斜杠	辅助输出特殊字符
()	圆括号	组织数据列表和调用函数列表
[]	中括号	数组索引，超级右中括号
{}	大括号	使用 progn 时组织一系列表达式
'	单引号	单引号中的表达式不会被解析
"	双引号	字符串分界符
,	逗号	列表中可选的元素分界符；在反引号作用范围内使用强制表达式解析
;	分号	行注释
:	冒号	位分隔符，范围操作
.	点号	getq 操作
+, -, *, /	算术运算	算术运算符；同时"/*"和"*/"组合也用于多行注释
!, ^, &,	逻辑运算	逻辑运算符
<, >, =	关系运算	比较运算，赋值以及位运算
#	#	用在第一列表示信号的特别解析
@	@	用在第一个字符表示保留字；另外在反引号作用范围内使用将强制表达式解析
?	问号	如果是第一个字符表示关键字
`	反引号	引号内的表达式不被解析
%	百分号	
\$	—	保留为将来使用

[注]：输出这些特殊字符都需要在字符前加“\”。

- 注释
 - 行注释方法，使用分号“;”，注释范围在分号当前行。
 - 多行注释，使用“/*”和“*/”对，注释范围在这这对符号内。
- 空格, 空格的限制比较少，只是要注意下面几个情况
 - 函数名和跟在后面的圆括号之间不能有空格，如 max(4 2) 是对的，而 max (4 2) 则是错的；
 - 表示负数的符号“-”和数字之间不能有空格，如-5，-(a*b) 有效，而- 5，- (a*b) 无效；
 - 表示算术运算的“+ - * /”在 2 个数字之间的位置要对称，如 a - b，a-b 有效，而 a -b 无效。
- 圆括号
 - 圆括号在 skill 中用于函数调用，分隔多个表达式，控制表达式的解析顺序
 - skill 中调用函数有 2 种方式，max(a b) 和 (max a b) 都有效。
- 超级右中括号，不推荐使用
 - 在使用了很多“(”的表达式后，可以用“]”来代表很多个用于结尾的“)”。

如 $f1(f2(f3(f4(x))))$ 等效于 $f1(f2(f3(f4(x))))$ 。

- 反引号，逗号和逗号加@

示例 1.

```
y = 1
'(x y z) => (x y z) ; 反引号使得表达式不被解析，所以其中的
y 不会被 1 代替
'(x , y z) => (x 1 z) ; 增加一个逗号在反引号表达式中，逗号后
面的表达式将被解析
```

示例 2.

```
x = 1
y = '(a b c)
'( , x , y z) => (1 (a b c) z) ; 逗号后的表达式都被解析了
'( , x , @y z) => (1 a b c z) ; ",@"的作用和逗号一样，只是被解
析对象是列表，且解析出来的元素代替了原来的列表
```

- 续行符“\”
 - 使用“\”可以让在一行的文字多行表示， 如

```
string = "This is \
a test." => "This is a test."
```

- 初始化列表的最大长度 6000，但是在程序运行中列表的长度无限制。

文件系统接口

- 路径
 - 绝对路径：Unix 中以“/”开始的路径，Windows 中以磁盘“C:”，“D:”，...”开始的路径
 - 相对路径，不是绝对路径的情况就是相对路径
 - 以“~/”开始的路径表示被查询的路径是用户的 **home** 文件夹；
 - 以“~username”开始的路径，如果 username 正好是一个用户的名字，那么被查询的路径将是该用户的 **home** 文件夹；
 - 以“./”开始的路径表示被查询的位置是当前的工作目录
 - 以“../”开始的路径表示被查询的位置为当前工作目录的**父文件夹**
 - Skill 路径
 - skill 支持用户定义若干不同的路径来存放 skill 源文件，并通过 skill 的路径管理函数来设置各个路径的访问顺序。如果同一个文件在不同路径下都存在，按照先到先得原则，即使用第一个被找到的文件。

- 设置 skill 查询路径的函数 setSkillPath, 如
setSkillPath(".", "~" "C:/Skill"), 就设置了 3 个 skill 查询路径, 先后顺序分别为 "."—当前工作路径, "~"—home 文件夹, "C:/Skill"—用户自定义文件夹。
- 获取 skill 查询路径列表的函数 getSkillPath, 如
setSkillPath(".", "~" "C:/Skill");
getSkillPath()=>(".", "~" "C:/Skill")
- 安装路径
 - 获取安装路径函数 getInstallPath, 如
getInstallPath()=>"C:/Cadence/Psd16.2"
- 检测文件状态
 - 检测文件是否存在函数 isFile, isFileName
 - 检测文件夹是否存在函数 isDir
 - 检测文件或文件夹的是否可以被读取的函数 isReadable
 - 检测文件或文件夹的是否可以被写入的函数 isWritable
- 路径操作
 - 生成和删除文件夹
 - createDir(arg), 如 createDir("c:/temp/skill"), 将在 c:/temp 下创建一个 skill 文件夹;
 - deleteDir(arg), 如 deleteDir("c:/temp/skill"), 将删除这个 skill 文件夹, 如果这是个空文件夹, 且用户具有删除它的权限。
 - 删除文件函数 deleteFile
 - 获取指定文件夹下所有文件名函数 getDirFiles
 - 获取当前工作目录 getWorkingDir
 - 改变工作目录 changeWorkingDir

端口

所有数据的输入输出都是经由一个被称为端口的东西实现的。

预定义的标准端口

端口名	作用
piport	标准输入端口, 等同于 C 中的 stdin
poport	标准输出端口, 等同于 C 中的 stdout
errport	错误输出端口, 等同于 C 中的 stderr
ptport	跟踪信息端口, 等同于 C 中的 stderr

- 打开端口
 - 打开一个文件读取端口 infile

infile("~/test/input.il")=> port:"~/test/input.il"

- - 打开一个文件输出端口 outfile

```
p = outfile("~/test/out.il" "w") => port:"~/test/out.il"
```

- 关闭端口 close

```
p = outfile("~/test/out.il" "w") => port:"~/test/out.il"
close(p)
```

I/O

- 输出
 - 非格式化输出

```
print("hello")      ; 输出 "hello"
println("hello")    ; 输出 "hello"加换行
```

- - 格式化输出

常见格式化输出规范

格式	参数类型	输出结果
%d	fixnum	十进制整数
%o	fixnum	八进制整数
%x	fixnum	十六进制整数
%f	flonum	浮点数，格式为[-]ddd.ddd
%e	flonum	浮点数，格式为[-]d.ddde[-]ddd
%g	flonum	不改变浮点数精度的情况下，以最少的空间表示
%s	string, symbol	输出字符串
%c	string, symbol	输出第一个字符
%n	fixnum, flonum	数字
%L	list	数据的默认形式输出
%P	list	点 Point
%B	list	Box



示例

```
printf( "hello, %s" "Willy" )=> "hello, Willy"
printf( "%d == %x" 20 20 )=> "20 == 14" ;把一个十进制书转换成 16
sprintf( sMsg "hello, %s" "You" )=> sMsg="hello, You"
fprintf( port "1st line, %d words" 10 ); 输出"1st line, 10 words"到文件
```

`printf("file is \"%s\" \"c:\\text.il\");` 输出 `file is "c:\text.il"`, 这里带了双引号

- 输入

输入函数列表

输入源 版)	操作函数	读取函数(评估版)	读取函数(未评估)
 File 文件	<code>load/loadi</code>		<code>lineread</code>
 String 字符串	<code>evalstring/loadstring/errsetstring</code>		<code>linereadstring</code>
	<code>instring/gets/getc/fscanf/close</code>		

文件输入输出示例

- 创建文件并输出数据到该文件的函数

```

procedure( DataOut()
  lData = list(
    ' ("just" 10 5.5)
    ' ("for" 20 10.2)
    ' ("test" 30 15.0)
    ' ("use" 40 17.8)
    ' ("done" 50 1.2)
  ); 定义数据列表 lData
  fDataOut = strcat( getTempDir() "./" "data.txt" ); 输出文件名和路
径, %temp%/data.txt.
  pDataOut = outfile( fDataOut "w" ); 建立输出端口
  foreach( item lData
    fprintf( pDataOut "%s %d %0.2f\n"          ; 输出数据到文件
    car(item) cadr(item) caddr(item) ; item=>每一个在
lData 中的列表
  ); end fprintf
); end foreach
close( pDataOut ) ; 关闭端口
printf( "I- output data from lData list to file %s, done!" fDataOut );
输出提示信息
); end procedure

```

- 从指定文件读取数据

```

procedure( DataIn()
  prog( (fDataIn pDataIn lData sWord nNum fNum) ;设置局部变量
    fDataIn = strcat( getTempDir() "./" "data.txt") ; 输入文件
    unless( isFile( fDataIn ) ; 如果文件不存在则跳转到函数末尾, 结
束函数
      printf("E- %s does not exist, double check please!" fDataIn)
      go(SkipToEnd) ; 在 prog 中支持 go 操作
    )
    pDataIn = infile( fDataIn ); 建立输入端口
    lData = nil 初始化列表变量
    while( fscanf( pDataIn "%s %d %0.2f" sWord nNum fNum ) ; 按照数
据文件格式读取
      lData = cons( list( sWord nNum fNum ) lData )
    ); end while
    ; 读取完毕 lData 里的数据将和上例中的一样
    close( pDataIn )
    SkipToEnd
  ); end prog
); end procedure

```

数字

- 整数

进制列表： 前缀 示例[十进制值]

二进制	0b 或 0B	0b1010 [10]
八进制	0	012 [10]
十进制		10
十六进制	0x 或 0X	0x11 [17]

- 浮点数

和其它的编程语言一样，如 1.0, 1., 0.5, .5, 1e6……

- 比例因子

比例因子			
字符	全名	比例	示例
Y	Yotta	1.00E+24	10Y [10e+25]
Z	Zetta	1.00E+21	10Z [10e+22]
E	Exa	1.00E+18	10E [10e+19]
P	Peta	1.00E+15	10P [10e+16]

T	Tera	1.00E+12	10T [10e+13]
G	Giga	1.00E+09	10G [10e+10]
M	Mega	1.00E+06	10M [10e+7]
k or K	kilo	1.00E+03	10k [10e+4]
%	percent	1.00E-02	5% [0.05]
m	milli	1.00E-03	5m [5.0e-3]
u	micro	1.00E-06	1.2u [1.2e-6]
n	nano	1.00E-09	1.2n [1.2e-9]
p	pico	1.00E-12	1.2p [1.2e-12]
f	femto	1.00E-15	1.2f [1.2e-15]
a	atto	1.00E-18	1.2a [1.2e-18]
z	zepto	1.00E-21	1.2z [1.2e-21]
y	yocto	1.00E-24	1.2y [1.2e-24]

- 常用的 skill 函数

通常整数和浮点数在一个表达式中计算的时候，结果默认是浮点数。

- - 判断数字类型

numberp(data)，判断 data 是不是数字类型；

fixp(data)，判断 data 是不是整型数；

floatp(data)，判断 data 是不是浮点数；

示例： numberp(5) => t,
 fixp(5) => t, 和 integer(5) 一样
 floatp(5) => nil,

(t, nil 分别代表真值和假值)。

- - 浮点数、整数转换

round(data)，转换浮点数到最近的整数；

fix(data)，转换浮点数到不大于它的最大的整数

float(data)，转换整数到相等大小的浮点数

示例： round(1.8) => 2, round(1.4) => 1
 fix(1.8) => 1, fix(1.4) => 1
 float(2) => 2.0

- - 字符串转换成数字

atoi(data), 转换字符串到对应的整数
atof(data), 转换字符串到对应的浮点数

示例: atoi(“3s”) => 3,
atof(“3s”) => 3.0

- - 其它数学运算函数

abs, cos, atan, log...具体请参考 skill 函数参考书 sklangref。

字符

skill 中没有一个专门的类型叫 char, 字符通常都是用 symbol 类型代表, 如 type(‘A’) => symbol。(skill 中的 symbol 相当于 C 语言中的变量)
可以使用 ASCII 来申明一个字符, 比如 char=’ \120, char=>p。
一些特殊的字符(转义序列)

new-line (line feed)	\n
horizontal tab	\t
vertical tab	\v
backspace	\b
carriage return	\r
form feed	\f
backslash	\\
double quote	\"
ASCII code ddd (octal)	\ddd

字符串

- 字符串定义

字符串是字符序列, 比如” 123”, “abs” ...。字符串用双引号来定义, 就像 C 语言一样。

可以直接打印的字符(处双引号本身)都可以直接被直接定义成字符串, 如”a”, “01”。

需要借助**转义字符(\)**才可以打印的字符, 以及双引号被定义成字符串时都需要和转义字符一起, 如”\””, “\\” 分别定义了字符串” 和\。

- 字符串限制

直接赋值的字符串长度不能超过 8191，但是在程序运行过程中的字符串长度没有限制。

- 常用函数
 - 判断字符串类型

stringp(data)，判断是否为字符串；

- - 合成，分开，提取

strcat(s_string1 s_string2 ...), 合成所有提供的字符串生成一个新的字符串

substring(s_string index length), 从指定位置提取特定长度的字符串

parseString(s_string breakchar), 用特定字符将原字符串断开

buildString(l_string breakchar), 用特定的字符将给定的字符串列表合成字符串

示例 strcat(“a” “1 b” “ 3 c”) => “a1 b 3 c” ;
substring(“abcdef” 3 2) => “cd” ;
parseString(“a-bc-def” “-”) => (“a” “bc” “def”)
buildString(‘(“1” “23” “4” “5a”) “+”) => “1+23+4+5a”

- - 字符串长度、比较

strlen(s_string)，得到字符串长度

strcmp(s_string1 s_string2)，比较 2 个字符串的大小

equal(s_string1 s_string2)，判断 2 个字符串是否相等

示例 strlen(“abc”) => 3
strcmp(“abc” “abb”) => 1, strcmp(“abc” “abc”)
=> 0
equal(“abc” “abb”) => nil

- - 输出与格式化

println(“hello”)=>” hello”，输出 hello；

printf(“float num is %.2f” 10.2)>” float num is 10.20。”

- - 其它

symbolToString, timeToString…… 参考 skill 函数文档

逻辑值

- t 表示逻辑真值
- nil 表示逻辑假值

数组 array

- 申明一个数组

申明数组需要用到关键字 declare，如 `declare(week[7])`，申明了一个能包含 7 个元素的数组 week。数组元素从 `week[0]` 到 `week[7-1]` 也即 `week[6]`。

- 赋值与读取

给数组的元素赋值，如 `week[0]="Mon"`；读取数组的一个元素，如 `week[0] => "Mon"`

- 特殊性

skill 中数组不同于其它语言的一点就是数组中的元素可以不是同一种类型，也即数组中可以既有字符串，又有数字等等。

- 常用的函数

`arrayp`，判断是否为数组类型。

列表 list

skill 中最常用到的一种类型就是列表，列表是一系列元素的结合，元素不必是同一个类型。比如 `(1 "as" "1")` 就是一个包含了三个元素 1、“as”和“1”的列表，而且这列表中这三个元素的位置上固定了的，比如说第二个元素就一定是“as”。

- 申明列表

申明列表有 2 种方式一种使用关键字 `list`，如 `a=list("ab" "c" "de")`，另一种用单引号 `' '`，如 `a='("ab" "c" "de")`，这 2 种方式定义的列表是一样的，这里都是定义了一个包含 3 个元素的列表。

- 常用操作列表函数
 - 访问列表元素

`car(list)`, `cdr(list)`, 分别获取列表的第一个元素和列表除第一个元素以外的元素列表

`car` 和 `cdr` 可以组合在一起使用, 如 `cadr(list)` \Rightarrow `car(cdr(list))`, 得到列表的第二个元素

`nth(index list)`, 获取指定位置的列表元素 `nth(0 list)` \Rightarrow `car(list)`

`member(item list)`, 查询列表中是否包含指定元素

```
示例 a=' ( 1 3 5 7 9 )
car(a) => 1
cdr(a) => ' (3 5 7 9 )
cadr(a) => 3
nth(4 a) => 9
member( 5 a ) => ' (5 7 9)
member( 4 a ) => nil
```

- - 添加和移除元素

`cons(item list)`, 将单个元素插入到列表的最前面成为列表的第一个元素;

`append1(item list)`, 和 `cons` 正好相反, 将元素插入到列表的最末尾;

`append(list1 list2)`, 合并 2 个列表为一个;

`remove(item list)`, 从列表中删除指定的元素。

```
示例 a=' (1 3 5 7 9)
cons( 2 a )=>' (2 1 3 5 7 9)
append1( a 2 ) => ' (1 3 5 7 9 2 )
append( a ' ("a" "bc"))=>' (1 3 5 7 9 "a" "bc")
remove( 5 a )=>' ( 1 3 7 9 )
```

- - 列表元素排序

`sort(list comparefunc)`, 根据比较函数来给列表排序, 比较函数可以自己定义;

`reverse(list)`, 反向排列列表

```
示例 a=' (5 3 7 0 9)
sort( a 'lessp )=>' (0 3 5 7 9)
reverse(a)>' (9 0 7 3 5)
```

- - 其它

`length`, `sortcar`, `tableToList`, `listp`, `last`...请参考 skill 函数文档

- 关联列表

列表中的每个元素都是一个子列表，且这些子列表由 2 个元素组成，也就是所谓的键值对，这样的列表称为关联列表。如' (("A" 1) ("B" 2) ("C" 3))'，参考函数 assoc。

关联表格 table

关联表格和数组类似，每一个表格相对应一个数值，同样在一个表格里面的数据类型可以是不一样的，不同的是数组的索引值是数字如 declare(week[7])，week[0]=" Mon"，而列表的索引值可以是任何值，如 table = makeTable("new table")，table[1]=" hello"，table["yes"]=" sure"。

- 申明表格

申明表格使用关键字 makeTable，如 table1=makeTable("hello" 1)，定义了表格 table，这个表格的名字是"hello"，所有未初始化的表格元素的值都是 1，table1[1]=>1， table1["a"]=1。

- 常用函数

tableToList(table)，转换表格为列表形式，如 tableToList(table1)=>' ((1 1) ("a" 1))
tablep(table)，判断是否为表格类型

Cadence 数据库数据 dbid

dbid 及 database id，就是数据库中的标识，每个 Cadence 软件平台中的数据都有这样的一个 dbid，skill 函数就是通过这些 id 来操作数据库文件。比如说 PCB Layout 工具 allegro，每一个在一个设计中的线路，电阻，网络都各自有个 dbid，通过相应的函数可以直接读取到这个 id。通过这个 id 可以得到这个 id 的相关信息，也可以得到相关信息的 dbid...

定义函数

Skill 中常用的 2 个定义函数的关键字是 procedure，lambda。

- procedure

procedure 的基本语法，

procedure(funcname(args)

```
) expX...
```

示例

```
procedure( trAdd( x y ) ;定义了包含 2 个参数的函数 trAdd
  printf( "Adding %d and %d ... %d \n" x y x+y )
  x+y ;默认的返回值
)
=> trAdd
trAdd( 6 7 ) => 13 ;函数名 trAdd, 2 个参数 6 和 7
```

- lambda

lambda 的基本语法,

```
trAddWithMessageFun = lambda( ( x y )
  printf( "Adding %d and %d ... %d \n" x y x+y )
  x+y
) => funobj:0x1814b90
```

lambda 定义的函数没有函数名, 所以将其赋值给一个变量。使用 apply 调用一个 lambda 函数, apply(trAddWithMessageFun ' (4 5)) => 9

局部和全局变量

- 定义局部变量

定义局部变量的 2 个关键字 let 和 prog。let 和 prog 的区别在于函数的返回值, let 的返回值是 let() 中的最后一句表达式, prog 的返回值比较多样, 还包括 return 命令的返回值。相对于 let, prog 还多了项功能, 在 prog() 中多了一个循环功能, 支持 go 命令。

- - let

如 let((args) expX...), 在 let 中的变量 args 在 let() 范围有效, 在 let() 范围外没有意义, 不存在。

- - prog

如 prog((args) expX...), 在 prog 中的变量 args 在 prog() 范围有效, 在 prog() 范围外没有意义, 不存在。

- - 定义全局变量

不在 prog 和 let 里面指定定义变量的地方定义的变量都是全局变量，但是请尽量少使用全局变量。关键字 **defvar** 可以用来定义全局变量，如 defvar(globalvar)。

返回值

如上节所示，使用 let 或不使用 let 和 prog 的函数返回值都是函数中最后的表达式，使用 prog 的情况下，会增加因为 return 语句带来的返回情况。

输入参数定义

输入参数有 2 种必然情况和 3 中可选情况

- 必然
 - 不含参数的函数

如 procedure(hello() println(“hello, world!”))
运行它 hello=>” hello,world!”

- - 含参数的函数

如 procedure(hello(arg)
printf(“hello, how are you %s” arg)
)

运行它 hello=>函数调用错误，必须提供一个参数
hello(“Emily”)=>” hello,how are you Emily”

- 可选, 有 3 种可选的情况, @rest, @optional, @key
 - @rest

@rest 选项允许任意数量的参数以列表的形式传入函数，跟在@rest 后面的参数的名字是任意的，请参考下例

```
procedure( trTrace( fun @rest args )
  let( ( result )
    printf( "\nCalling %s passing %L" fun args )
    result = apply( fun args )
    printf( "\nReturning from %s with %L\n" fun result )
```

```

    result
  ) ; let
) ; procedure

```

调用函数 `trTrace('plus 1 2 3) => 6`，其实是运行了 `apply(plus 1 2 3)`。

-
- @optional

@optional 选项允许设置一些有默认值的输入参数，如果调用函数的过程中提供了这些参数的值，新值将替换默认值。

```

procedure( trBuildBBox( height width
  @optional ( xCoord 0 ) ( yCoord 0 ) )
  list(
    xCoord:yCoord ;;; lower left
    xCoord+width:yCoord+height ;;; upper right
  )
) ; procedure

```

在这个示例中 xCoord 和 yCoord 的默认值都是 0。运行

```

trBuildBBox( 1 2 ) => ((0 0) (2 1)) ; xCoord, yCoord 值保持 0
trBuildBBox( 1 2 4 ) => ((4 0) (6 1)) ; xCoord 值改为 4, yCoord 值保持 0
trBuildBBox( 1 2 4 10 ) => ((4 10) (6 11)) ; xCoord 值改为 4, yCoord 值改为 10

```

-
- @key

@key 作用和@optional 相同，只是@key 中的参数可以以任意的顺序调用。

```

procedure( trBuildBBox(
  @key ( height 0 ) ( width 0 ) ( xCoord 0 ) ( yCoord 0 ) )
  list(
    xCoord:yCoord ;;; lower left
    xCoord+width:yCoord+height ;;; upper right
  )
) ; procedure

```

运行

```

trBuildBBox() => ((0 0) (0 0)) ; 什么默认值都不改变
trBuildBBox( ?height 10 ) => ((0 0) (0 10)) ; 只改变 height 的值
trBuildBBox( ?width 5 ?xCoord 10 ) => ((10 0) (15 0)); 改变 width 和 xCoord

```

4.5 类型检查 Skill 提供了函数输入参数类型的检查的功能。

字符	含义
S	Symbol or string
n	Number (fixnum, flonum)
u	Function - either the name of a function(symbol) or a lambda function body (list)
g	Any data type

示例 `procedure(f(x y "nn") x**2 + y**2)`, nn 表示函数 f 需要 2 个数字类型的参数输入

函数的限制

函数的几个限制

- 必须的参数数量小于 255
- 可选的参数数量少于 255
- 在 let, prog 中使用的参数数量少于 255
- 一个函数所能容纳的参数数量不多于 32k

示例

- go 在 prog 中的应用

```
procedure( guessNum(arg)
  prog(
    unless( numberp(arg)
      println("E- input a number please!")
      go( SkipToEnd ) ;如果输入类型不对, 程序将直接跳转到 SkipToEnd
    )
    if( fixp( arg ) then
      println( "input is a fix type number" )
    else
      println( "input is a float type number" )
    ); end if
    SkipToEnd
  ); end prog
); end procedure
```

- lambda 的运用


```
signalList = ' ( "abc" "d" "abcdef" "ww" "cde" )  
sort( signalList  
      'lambda( ( a b ) strlen(a) <= strlen(b) )  
      ; lambda 这里定义了一个字符长度比  
      较函数  
); 输出结果为 “(“d” “ww” “abc” “cde” “abcdef”)
```