Gura ライブラリリファレンス

Updated: October 25, 2013

Official site: http://www.gura-lang.org/

目次

1. このリファレンスについて

本リファレンスは Gura の本体や標準添付のモジュールで定義されている関数やクラスの仕様について説明します。Gura 言語そのものの仕様などについては「**Gura 言語マニュアル**」を参照してください。

また、仕様の大きなモジュールについては、以下のように独立したリファレンスが用意されています。

- **Gura モジュールリファレンス** – cairo
- **Gura モジュールリファレンス** – opengl
- **Gura モジュールリファレンス** – sdl
- **Gura モジュールリファレンス** – tk
- **Gura モジュールリファレンス** – wx

2. 組込み関数

2.1. テキスト表示

`print(value*):map:void`

引数 `value` の値を文字列に変換した結果を連結して標準出力に出力します。

`println(value*):map:void`

引数 `value` の値を文字列に変換した結果を連結して標準出力に出力し、最後に改行します。

`printf(format:string, values*):map:void`

文字列 `format` 中のフォーマッタ指定に基づいてリストの内容を文字列に変換します。書式の形式は `%[flags][width][.precision]specifier` のようになります。

`[specifier]` には以下のうちのひとつを指定します。

specifier	説明
d, i	10 進符号つき整数
u	10 進符符号なし整数
b	2 進数整数値
o	8 進符号なし整数
x, X	16 進符号なし整数
e, E	指数形式浮動小数点数 (E は大文字で出力)
f, F	小数形式浮動小数点数 (F は大文字で出力)
g, G	eまたはf形式の適した方 (G は大文字で出力)
s	文字列
c	文字

`[flags]` には以下のうちのひとつを指定します。

flags	説明
+	プラス数値のとき、先頭に + 記号をつけます
-	左詰めで文字列を配置します
(空白)	プラス数値のとき、先頭に空白文字をつけます
#	2 進、8 進、16 進整数の変換結果に対しそれぞれ "0b", "0", "0x" を先頭につけます
0	桁数の満たない部分を 0 で埋めます

`[width]` には最小の文字幅を 10 進数値で指定します。文字列に変換した結果の長さがこの数値に満たないとき、残りの幅を空白文字 (文字コード 32) で埋めます。長さがこの数値以上の場合は何もしません。`[width]` の位置に数値ではなくアスタリスク "*" を指定すると、最小の文字幅を指定する数値を引数から取得します。

`[precision]` は specifier によって意味が異なります。浮動小数点数に対しては、小数点以下の表示桁数の指定になります。

2.2. 制御構文

2.2.1. Gura における制御構文

Gura は言語仕様の中に制御構文というものを持っていません。繰り返しや条件分岐などはすべて関数呼び出しで実現しています。これら関数の名前や引数などを既存言語の制御構文と似せているので、動作内容が類推しやすくなっています。

2.2.2. 関数内のフロー制御

`return (value?):symbol_func`

関数の処理を中断し、呼び出し元のフローに戻ります。引数として`value`を渡すと、中断した関数の評価値をその値に設定します。省略すると、評価値は `nil` になります。

この関数は、アトリビュート `:symbol_func` が指定されています。つまり、引数が必要ない場合は引数リストの括弧を省略して呼び出すことができます。

2.2.3. 繰り返し

`repeat (n?:number) {block}`

引数で指定した回数だけ `block` の処理を繰り返します。引数は省略可能で、省略した場合無限ループになります。

`while (`cond) {block}`

引数で指定した式が条件を満たす間だけ `block` の処理を繰り返します。

`for (`expr+) {block}`

一つ以上のイテレータ代入式を引数にとり、イテレータが終了するまで `block` の処理を繰り返します。イテレータ代入式の形式は以下のようになります。

symbol in *iterator*

[*symbol1*, *symgol2* ..] in *iterator*

最初の形式では、イテレータの要素が *symbol* で表される変数に代入されます。もし要素がリストであれば、*symbol* に代入される値はそのリストそのものになります。二番目の形式では、イテレータの要素がリストであればリストの要素ごとに対応する位置にあるシンボルの変数に値を代入します。要素がリストでない場合、全てのシンボルの変数に同じ値が代入されます。

イテレータ代入式が二つ以上指定された場合、一回のループで引数中のイテレータを一つずつ評価していきます。こうして、いずれかのイテレータが終了するまで処理が繰り返されます。つまり、イテレータの要素数が異なるときは、ループの回数は一番短いイテレータの要素数にあわせられます。

`cross (`expr+) {block}`

一つ以上のイテレータ代入式を引数にとり、イテレータが終了するまで `block` の処理を繰り返します。イテレータ代入式が一つのとき、処理内容は `for` 関数に一つの引数を渡したときと同じです。二つのイテレータ代入式を指定すると多重ループになり、一つ目のイテレータが外側、二つ目のイテレータが内側のループを構成します。イテレータ代入式を複数指定することも可能で、n 個の代入式を指定するとn重の

多重ループになります。

スクリプト	実行結果

`cross (x in 0..1, y in 0..2) {`
`    printf('[%d,%d]', x, y)`
`}`
	`[0,0][0,1][0,2][1,0][1,1][1,2]`

2.2.4. 繰り返し中のフロー制御

`break(value?):symbol_func`

繰り返し関数の処理を中断します。引数として value を渡すと、中断した繰り返し関数の戻り値をその値に設定します。省略すると、繰り返し関数の戻り値は nil になります。

この関数は、アトリビュート :symbol_func が指定されています。つまり、引数が必要ない場合は引数リストの括弧を省略して呼び出すことができます。

`continue(value?):symbol_func`

繰り返し処理の続きをスキップして先頭に戻ります。引数として value を渡すと、ループのその回の評価値をその値に設定します。省略すると、その回の評価値は nil になります。

この関数は、アトリビュート :symbol_func が指定されています。つまり、引数が必要ない場合は引数リストの括弧を省略して呼び出すことができます。

2.2.5. 条件分岐

`if (`cond):leader {block}`

条件 cond が true のとき、block の内容を実行します。false のとき、このあとに elsif または else 関数が連結されていると、それらを評価します。

`elsif (`cond):leader:trailer {block}`

if または elsif 関数の後に連結して使用します。条件 cond が true のとき、block の内容を実行します。false のとき、このあとに elsif または else 関数が連結されていると、それらを評価します。

`else():trailer {block}`

if または elsif 関数の後に連結して使用します。無条件で block の内容を実行します。

条件分岐のスクリプト例を以下に示します。

スクリプト

`if (x == 0) {`
`    println('x value is zero')`
`} elsif (x == 1) {`
`    println('x value is one')`
`} else {`
`    println('other case')`
`}`

2.2.6. 例外処理

`try():leader {block}`

block を実行し、その間に例外が発生すると、後に連結された catch 関数を実行します。

`catch(errors*:error):leader:trailer {block}`

try または catch 関数の後に連結して使用します。

発生した例外が引数 errors のいずれかに合致する場合 block を実行し、ブロックパラメータを |error:error| という形式で渡します。error は検出したエラーに対応するerror 型のインスタンスです。

例外が引数に合致しない場合、後に連結された exept 関数を実行します。引数 errors を指定しないと、すべての例外に合致します。

`raise(error:error, msg:string => 'error', value?)`

例外を発生します。引数 error にエラーインスタンス、msg にエラーメッセージを指定します。引数 value にはエラーの追加情報を指定します。

例外処理のスクリプト例を以下に示します。

スクリプト

```
try {
    // some jobs
} catch(ValueError) {|e|
    println('ValueError captured: ', e.text)
} catch(IOError) {|e|
    println('IOError captured: ', e.text)
} catch {|e|
    println('other error captured: ', e.text)
}
```

2.2.7. switch 文

`switch() {block}`

switch 文を構成します。block 中は case または default 関数の呼び出しを記述します。

`case(`cond) {block}`

条件 cond が true のとき、block の内容を実行し、switch 関数を抜けます。false のとき、switch の次に記述されている case や default 関数の呼び出しに移ります。

`default() {block}`

無条件に block の内容を実行し、switch 関数を抜けます。

2.3. データ変換

`chr(num:number):map`

UTF-8 文字コードを文字列に変換します。

`ord(str:string):map`

文字列の先頭の文字に対応する UTF-8 文字コードを返します。

`int(value):map`

数値を整数に変換した結果を返します。value が文字列のとき、これを数値に変換した結果を整数にして返します。

`tonumber(value):map:[nil,zero,raise,strict]`

文字列を number 型に変換した結果を返します。デフォルトでは、文字列の初めの部分が数値とみなせれば変換が成功します。アトリビュート :strict をつけると、文字列中に数値以外の文字が含まれていたとき変換に失敗するようになります。

変換に失敗したときのふるまいを以下のアトリビュートで指定することができます。

:nil　nil 値を返します (デフォルト)。

:zero　数値 0 を返します。

:raise ValueError 例外を発生します。

`tostring(value):map`

任意の値を文字列に変換した結果を返します。

`tosymbol(str:string):map`

文字列をシンボルに変換した結果を返します。

`hex(num:number, digits?:number):map:[upper]`

数値を 16 進文字列に変換した結果を返します。digits に最少の桁数を指定します。変換した結果の桁が digits にみたない場合、先頭を 0 で埋めます。アルファベットは小文字になりますが、アトリビュート:upper を指定すると大文字になります。

2.4. クラス操作

`class(superclass?:function) {block?}`

クラスを生成します。詳細は「**Gura 言語マニュアル**」を参照ください。

`struct(`args+):[loose] {block?}`

構造体のコンストラクタ関数を生成します。詳細は「**Gura 言語マニュアル**」を参照ください。

`classref(type+:expr):map {block?}`

指定の型のクラスへの参照を返します。

`super(obj):map {block?}`

スーパークラスのメソッドや変数を参照するオブジェクトを返します。

2.5. 変数スコープ操作

`extern(`syms+)`

関数の内部で使用します。指定したシンボルを、関数の外部で宣言されている変数に対する参照に設定します。指定のシンボルが外部スコープでみつからない場合、エラーになります。

`local(`syms+)`

関数以外のブロックの内部で使用します。引数に指定したシンボルを、ブロックの内部のスコープに対する参照に設定します。

`scope(target?) {block}`

ローカルスコープを作成して block の内容を評価し、block で最後に評価された値を戻り値として返します。引数 target にモジュールまたは environment インスタンスを指定すると、それらのスコープ内で block の内容を評価します。

locals(module?:module)

現在のスコープにアクセスするenvironment 型データを返します。引数moduleを指定すると、そのモジュールにアクセスする environment 型データを返します。

outers()

現在のスコープのひとつ外のスコープにアクセスする environment 型データを返します。

undef(`value+):[raise]

引数valueで指定されているシンボルを未定義にします。未定義のシンボルに対してこの関数を実行すると、単に無視されます。未定義のシンボルを指定したときにエラーを起こさせるには、アトリビュート :raise を指定します。

2.6. イテレータ生成

fill(n:number, value?) {block?}

引数 n で指定した数だけ同じ値 value を返すイテレータを生成します。引数 value を省略すると nil 値になります。

block をつけると、要素ごとにその内容を評価します。ブロックパラメータの形式は |value, idx:number| で、value に関数に渡した value の値、idx に 0 から始まるインデクス番号が入ります。

interval(a:number, b:number, samples:number):map:[open,open_l,open_r] {block?}

条件 $a \leq x \leq b$ を満たす x の数列を引数 samples で指定した数だけ等間隔で生成するイテレータを返します。アトリビュートを指定することで、条件を以下のように変えることができます。

:open_l	$a < x \leq b$
:open_r	$a \leq x < b$
:open	$a < x < b$ (アトリビュートに :open_l:open_r と指定したのと同じです)

block をつけると、要素ごとにその内容を評価します。ブロックパラメータの形式は |num:number, idx:number| で、num に生成した数値、idx に 0 から始まるインデクス番号が入ります。

range(num:number, num_end?:number, step?:number):map {block?}

引数 num のみを指定すると、0 から num-1 までの整数を生成するイテレータを返します。引数 num と num_endを指定すると、numからnum_end-1 までの整数を生成するイテレータを返します。引数stepで数値の間隔を指定します。省略すると間隔が 1 になります。

block をつけると、要素ごとにその内容を評価します。ブロックパラメータの形式は |num:number, idx:number| で、num に生成した数値、idx に 0 から始まるインデクス番号が入ります。

iterator(value+) {block?}

引数に指定したデータからイテレータを生成し、それを結合したイテレータを返します。

block をつけると、要素ごとにその内容を評価します。ブロックパラメータの形式は |value, idx:number| で、value に要素の値、idx に 0 から始まるインデクス番号が入ります。

2.7. フォーマット変換

format(format:string, values*):map

printf 関数のフォーマットでデータを文字列に変換します。フォーマット中に記述する指定子については printf の説明をご覧ください。

zipv(values+) {block?}

引数 values で指定した値をまとめたリストを生成します。values がすべてスカラーの場合、ひとつのリストを返します。values の中にリストまたはイテレータが含まれる場合、その要素ごとにリストに変換します。このとき、リストまたはイテレータが 2 つ以上含まれていると、そのうち最も要素数が少ない数だけリストに変換します。

2.8. モジュール

import(`module, `alias?):[overwrite,binary] {block?}

モジュールをインポートします。詳細は「Gura 言語マニュアル」を参照ください。

module() {block}

生成したローカルモジュールの中で block の内容を評価した後、そのモジュールへの参照を返します。

2.9. データ型チェック

istype(value, type+:expr):map

引数 value が type で表わされる型のデータのとき、true を返します。組み込みオブジェクトの型をチェックするために、以下のコンビニエンス関数が用意されています。

関数	等価な呼び出し
isbinary(value)	istype(value, `binary)
isboolean(value)	istype(value, `boolean)
isclass(value)	istype(value, `class)
iscomplex(value)	istype(value, `complex)
isdatetime(value)	istype(value, `datetime)
isdict(value)	istype(value, `dict)
isenvironment(value)	istype(value, `environment)
iserror(value)	istype(value, `error)
isexpr(value)	istype(value, `expr)
isfunction(value)	istype(value, `function)
isiterator(value)	istype(value, `iterator)
islist(value)	istype(value, `list)
ismatrix(value)	istype(value, `matrix)

ismodule(value)	istype(value, `module)
isnumber(value)	istype(value, `number)
issemaphore(value)	istype(value, `semaphore)
isstring(value)	istype(value, `string)
issymbol(value)	istype(value, `symbol)
istimedelta(value)	istype(value, `timedelta)
isuri(value)	istype(value, `uri)

isinstance(value, type+:expr):map

引数 value が type で表わされる型か、その派生クラスのデータのとき true を返します。

typename(`value)

引数 value が未定義のシンボルの場合、"undefined" を返します。それ以外の場合、value を評価し、その結果のデータ型を文字列で返します。

isdefined(`symbol)

引数 symbol が定義済みのシンボルの場合に true、未定義のときに false を返します。

2.10. 演算・統計

choose(index:number, values+):map

引数 values に 1 つ以上の値を列挙したとき、引数 index で指定した位置にある values の値を返します。例えば、choose(2, 'one', 'two', 'three') は 'three' を返します。

cond(flag:boolean, value1, value2):map

引数 flag が true のとき value1、false のとき value2 の値を返します。

max(values+):map

引数 values に列挙した値のうち、最大の値を返します。

min(values+):map

引数 values に列挙した値のうち、最少の値を返します。

mod(n, m):map

引数 n を m で割った余りを返します。

2.11. スクリプト評価

eval(expr:expr):map

引数 expr の内容を現在の環境で評価し、その結果を返します。

2.12. 乱数

randseed(seed:number)

乱数のシードを設定します。

`rand(range?:number)`

引数を指定しない場合、0 以上 1 未満の範囲で乱数を発生します。引数 `range` を指定すると、0 から (range – 1) までの整数を返します。range が整数でない場合、整数に丸められます。

`rands(range?:number, num?:number) {block?}`

引数 `num` で指定した数だけ乱数を発生するイテレータを返します。引数 `range` を指定しない場合、0 以上 1 未満の範囲で乱数を発生します。引数 `range` を指定すると、0 から (range – 1) までの整数を返します。`range` が整数でない場合、整数に丸められます。

`block` をつけると、ひとつの乱数ごとにその内容を評価します。ブロックパラメータの形式は `|num:number, idx:number|` で、`num` に生成した乱数、`idx` に 0 から始まるインデクス番号が入ります。

2.13. その他

`dir(obj?):[noesc]`

引数 `obj` に属している関数や変数のシンボルをリストで返します。`obj` の種類によってシンボルの内容は以下のようになります。

obj の種類	シンボルの内容
モジュール	モジュール内の関数、変数
クラス (コンストラクタ関数)	メソッド、プロパティ
オブジェクト	メソッド、プロパティ

アトリビュート `:noesc` をつけると、`obj` の種類がクラスの場合、派生元のメソッドおよびプロパティは除外します。

`help(func:function):map:void`

関数 `func` のヘルプを標準出力に表示します。

3. 定義済み変数

変数	内容
root	トップレベルスコープの environment インスタンスを返します。 モジュール内からトップレベルスコープに変数や関数を追加するときに参照します。

4. 組込みクラス

4.1. object クラス

4.1.1. 概要

すべてのオブジェクトの基本クラスになるクラスです。

4.1.2. インスタンスの生成

object()

object 型インスタンスを生成します。

4.2. complex クラス

4.2.1. 概要

複素数を扱うクラスです。

4.2.2. インスタンスメソッド

complex#abs()

複素数の絶対値を返します。

complex#arg():[deg]

複素数の偏角をラジアン値で返します。アトリビュート:deg をつけると、degree 値で返します。

complex#imag()

複素数の虚数成分を返します。

complex#norm()

複素数のノルム値を返します。

complex#real()

複素数の実数成分を返します。

4.3. fraction クラス

4.3.1. 概要

分数を扱うクラスです。

4.3.2. インスタンスメソッド

fraction#denominator()

分母を返します。

fraction#numerator()

分子を返します。

`fraction#reduce()`

分子と分母を通分した結果を返します。

4.4. binary クラス

4.4.1. 概要

binary クラスは、バイナリデータを保持してするインスタンスを生成するクラスです。string クラスとよく似ていますが、string クラスのインスタンスで保持されるデータが UTF-8 エンコーディングされた文字データに限られ、操作も文字単位であることに対し、binary クラスは任意のバイナリデータを扱え、処理単位も 8bit 幅のデータになります。

また、stringクラスはインスタンスの内容を更新することができませんが、binaryクラスのインスタンスはデータを追加したり既存のデータを書き換えることができます。この特徴により、binary クラスのインスタンスを stream に変換して、ストリームデータの出力先として扱うことができます。

4.4.2. インスタンスの生成

コンストラクタ関数 binary を使ってインスタンスを生成します。

`binary(buff*)`

複数のデータを結合した結果を binary 型として返します。引数 buff には string 型または binary 型のデータを 0 個以上指定します。データを指定しない場合は、空の binary 型データを生成します。これは、バイナリリテラルで b'' と指定したのと同じです。string 型は UTF-8 エンコードの内部表現をそのままバイナリ列として結合します。

4.4.3. クラスメソッド

`binary.pack(format:string, value*):map`

引数 fomat で指定したフォーマットに基づいて、value の内容を埋め込んだバイナリデータを binary 型として返します。format 中には、データの個数を表す数値に続いて以下の指定子を記述します。

指定子	説明
x	データを埋め込まず、アドレスを指定のバイト数分だけ進めます。
c	string 型データをとり、文字列の最初の 1 バイトをバイナリ列に挿入します。
b	number 型データをとり、符号付きバイト数値としてバイナリ列に挿入します。
B	number 型データをとり、符号無しバイト数値としてバイナリ列に挿入します。
h	number 型データをとり、符号付き 2 バイト数値としてバイナリ列に挿入します。
H	number 型データをとり、符号無し 2 バイト数値としてバイナリ列に挿入します。
i	number 型データをとり、符号付き 4 バイト数値としてバイナリ列に挿入します。
I	number 型データをとり、符号無し 4 バイト数値としてバイナリ列に挿入します。
l	number 型データをとり、符号付き 4 バイト数値としてバイナリ列に挿入します。
L	number 型データをとり、符号無し 4 バイト数値としてバイナリ列に挿入します。

q	number 型データをとり、符号付き 8 バイト数値としてバイナリ列に挿入します。
Q	number 型データをとり、符号無し 8 バイト数値としてバイナリ列に挿入します。
f	number 型データをとり、float 数値（4 バイト） としてバイナリ列に挿入します。
d	number 型データをとり、double 数値（8 バイト） としてバイナリ列に挿入します。
s	string 型データをとり、指定の文字エンコードに変換してバイナリ列に挿入します。文字エンコード名は、format 中にブレース記号 "{" および "}" で囲んで指定します。この指定子の場合、先行する個数を表す数値は、変換した結果からバイナリ列に挿入するバイト数になります。

2 バイト、4 バイト、8 バイト数値のバイトオーダーは以下の指定子で変更できます。

指定子	説明
@	以降の数値フォーマットをシステム依存のエンディアンに設定します。
=	以降の数値フォーマットをシステム依存のエンディアンに設定します。
<	以降の数値フォーマットをリトルエンディアンに設定します。
>	以降の数値フォーマットをビッグエンディアンに設定します。
!	以降の数値フォーマットをビッグエンディアンに設定します。

データの個数として数値の代わりにアスタリスク記号 "*" を指定すると、引数から数値データをとりだし、それをデータの個数とします。

4.4.4. インスタンスメソッド

`binary#add(buff+:binary):map:reduce`

binary インスタンスに他の binary を追加します。

`binary#decode(codec:codec)`

binary の内容を codec で指定した文字コーデックを使ってデコードし、結果を string 型で返します。

`binary#dump():void:[upper]`

binary の内容を標準出力にダンプ表示します。アルファベットは小文字で表示されますが、アトリビュート :upper をつけると大文字になります。

`binary#each() {block?}`

binary の内容を 1 バイトずつとりだし、number 型で返すイテレータを生成します。

block をつけると、1 バイトとりだすごとにその内容を評価します。ブロックパラメータの形式は |num:number, idx:number| で、num にとりだしたバイト数値、idx に 0 から始まるインデクス番号が入ります。

`binary#encodeuri()`

URI 書式で処理ができるようにした文字列を返します。

`binary#len()`

バイト数を返します。

binary#pointer(offset:number => 0)

t.b.d.

binary#reader() {block?}

読み込み用ストリームに変換した結果を返します。

binary#store(offset:number, buff+:binary):map:reduce

binary インスタンスの、指定の位置に他の binary の内容を格納します。引数 offset はバイト単位で指定します。現在のサイズを超えたところに格納位置を指定すると、そこまでの範囲を 0 で埋めます。

binary#unpack(format:string, offset:number => 0)

引数 fomat で指定したフォーマットに基づいて、バイナリデータから数値や文字列を抽出し、その結果をリストで返します。引数 offset は、抽出する位置をバイト単位で指定します。指定した位置がバイナリデータの範囲外になるとエラーになります。

format 中には、データの個数を表す数値に続いて以下の指定子を記述します。

指定子	説明
x	抽出はせず、アドレスを指定のバイト数分だけ進めます。
c	1 バイトを抽出し、それを文字コードとした string 型データを返します。
b	1 バイトを抽出し、それを符号付きバイト数値とした number 型データを返します。
B	1 バイトを抽出し、それを符号無しバイト数値とした number 型データを返します。
h	2 バイトを抽出し、それを符号付き 2 バイト数値とした number 型データを返します。
H	2 バイトを抽出し、それを符号無し 2 バイト数値とした number 型データを返します。
i	4 バイトを抽出し、それを符号付き 4 バイト数値とした number 型データを返します。
I	4 バイトを抽出し、それを符号無し 4 バイト数値とした number 型データを返します。
l	4 バイトを抽出し、それを符号付き 4 バイト数値とした number 型データを返します。
L	4 バイトを抽出し、それを符号無し 4 バイト数値とした number 型データを返します。
q	8 バイトを抽出し、それを符号付き 8 バイト数値とした number 型データを返します。
Q	8 バイトを抽出し、それを符号無し 8 バイト数値とした number 型データを返します。
f	4 バイトを抽出し、それを float 数値とした number 型データを返します。
d	8 バイトを抽出し、それを double 数値とした number 型データを返します。
s	指定の文字エンコードで文字列に変換した結果を string 型データで返します。文字エンコード名は、format 中にブレース記号 "{" および "}" で囲んで指定します。この指定子の場合、先行する個数を表す数値は、変換した結果からバイナリ列から抽出するバイト数になります。

2 バイト、4 バイト、8 バイト数値のバイトオーダーは以下の指定子で変更できます。

指定子	説明
@	以降の数値フォーマットをシステム依存のエンディアンに設定します。
=	以降の数値フォーマットをシステム依存のエンディアンに設定します。

<	以降の数値フォーマットをリトルエンディアンに設定します。
>	以降の数値フォーマットをビッグエンディアンに設定します。
!	以降の数値フォーマットをビッグエンディアンに設定します。

binary#unpacks(format:string, offset:number => 0, cnt?:number) {block?}

引数 fomat で指定したフォーマットに基づいて、バイナリデータから数値や文字列を抽出するイテレータを返します。引数 format と offset の意味は binary#unpack と同じです。cnt は抽出する回数を指定し、これが省略されるとバイナリデータの終端まで抽出を続けます。

block をつけると、データを抽出するごとにその内容を評価します。ブロックパラメータの形式は |elems[], idx:number| で、elems に抽出結果、idx に 0 から始まるインデクス番号が入ります。

binary#writer() {block?}

書き込み用ストリームに変換した結果を返します。初期のオフセットは binary の終端に設定され、書きこんだデータは追記されていきます。

4.5. pointer クラス

4.5.1. 概要

pointer クラスは、binary インスタンス内の指定位置にあるデータにアクセスするためのクラスです。

4.5.2. インスタンスの生成

binary#pointer メソッドで生成します。

4.5.3. インスタンスメソッド

pointer#forward(distance:number):reduce

オフセットを指定数だけ進めます。

pointer#reset()

オフセットを 0 にします。

pointer#pack(format:string, value+):reduce:[stay]

引数 fomat で指定したフォーマットに基づいて、value の内容を pointer が現在指している binary 内に埋め込みます。format 内に記述する指定子は binary.pack を参照ください。

pointer のオフセットは抽出したデータ数だけ進みます。アトリビュート:stay を指定すると、現在の位置にとどまります。

pointer#unpack(format:string):[stay]

引数 fomat で指定したフォーマットに基づいて、pointer が現在指している binary 内のバイナリデータから数値や文字列を抽出し、その結果をリストで返します。指定した位置がバイナリデータの範囲外になるとエラーになります。

format 中に記述する指定子は binary.unpack を参照ください。

pointer のオフセットは抽出したデータ数だけ進みます。アトリビュート:stay を指定すると、現在の位置にとどまります。

pointer#unpacks(format:string, cnt?:number)

t.b.d

4.6. codec クラス

4.6.1. 概要

Gura の文字列の内部コードである UTF-8 と他のエンコーディングとで文字コードを変換するクラスです。

4.6.2. インスタンスの生成

コンストラクタ関数 codec を使ってインスタンスを生成します。

codec(encoding:string, process_eol:boolean => false)

指定したエンコーディング名に対応する codec 型インスタンスを返します。引数 encoding にエンコーディング名を指定します。対応する codec がない場合はエラーになります。process_eol は行末コードの変換の有無を表し、true を指定すると CR-LF コードと LF コードの変換を行います。false の場合はこの変換を行いません。

4.6.3. クラスプロパティ

プロパティ	型	R/W	説明
bom_utf8	binary	R	UTF8 の BOM (Byte Order Mark)
bom_utf16be	binary	R	UTF16BE の BOM
bom_utf16le	binary	R	UTF16LE の BOM
bom_utf32be	binary	R	UTF32BE の BOM
bom_utf32le	binary	R	UTF32LE の BOM

4.6.4. クラスメソッド

codec.dir()

利用可能な文字コーデックの名前の一覧をリストで返します。

4.6.5. インスタンスメソッド

codec#decode(buff:binary):map

引数 buff の内容をデコードした結果を string 型で返します。

codec#encode(string:string):map

引数 string の内容をエンコードした結果を binary 型で返します。

4.7. color クラス

4.7.1. 概要

赤・緑・青およびアルファ値から成る色データを表現するクラスです。

4.7.2. インスタンスの生成

`color(name, alpha?:number):map`

指定した名前に対応する color インスタンスを生成します。引数 name に、string 型または symbol 型で色の名前を指定します。色の名前は Web 標準カラー名および X11 色名称の中のひとつを選択します。引数 alpha にはアルファ値を 0 から 255 の間の数値で指定します。

`color(red:number, green:number, blue:number, alpha?:number):map`

指定した RGB 値を持つ color インスタンスを生成します。引数 red、green および blue に 0 から 255 の間の数値で RGB 値を指定します。引数 alpha にはアルファ値を 0 から 255 の間の数値で指定します。

4.7.3. Web 標準カラー名

Web 標準カラー名と RGB 値を以下にまとめます。

名前	RGB 値
black	0, 0, 0
maroon	128, 0, 0
green	0, 128, 0
olive	128, 128, 0
navy	0, 0, 128
purple	128, 0, 128
teal	0, 128, 128
gray	128, 128, 128

名前	RGB 値
silver	192, 192, 192
red	255, 0, 0
lime	0, 255, 0
yellow	255, 255, 0
blue	0, 0, 255
fuchsia	255, 0, 255
aqua	0, 255, 255
white	255, 255, 255

4.7.4. クラスプロパティ

プロパティ	型	R/W	説明
names	string	R	カラー名の一覧が格納されています
Black	color	R	色要素 #000000 を持った color インスタンスです
Maroon	color	R	色要素 #800000 を持った color インスタンスです
Green	color	R	色要素 #008000 を持った color インスタンスです
Olive	color	R	色要素 #808000 を持った color インスタンスです
Navy	color	R	色要素 #000080 を持った color インスタンスです
Purple	color	R	色要素 #800080 を持った color インスタンスです
Teal	color	R	色要素 #008080 を持った color インスタンスです
Gray	color	R	色要素 #808080 を持った color インスタンスです
Silver	color	R	色要素 #c0c0c0 を持った color インスタンスです
Red	color	R	色要素 #ff0000 を持った color インスタンスです
Lime	color	R	色要素 #00ff00 を持った color インスタンスです
Yellow	color	R	色要素 #ffff00 を持った color インスタンスです

Blue	color	R	色要素 #0000ff を持った color インスタンスです
Fuchsia	color	R	色要素 #ff00ff を持った color インスタンスです
Aqua	color	R	色要素 #00ffff を持った color インスタンスです
White	color	R	色要素 #ffffff を持った color インスタンスです

4.7.5. インスタンスプロパティ

プロパティ	型	R/W	説明
red	number	R/W	赤要素を 0 から 255 までの数値で表します
green	number	R/W	緑要素を 0 から 255 までの数値で表します
blue	number	R/W	青要素を 0 から 255 までの数値で表します
alpha	number	R/W	アルファ要素を 0 から 255 までの数値で表します
gray	number	R	グレー値を取得します。この値は、赤要素 R、緑要素 G および青要素 B の値をもとに以下の演算式で算出したものです。 0.299 * R + 0.587 * G + 0.114 * B

4.7.6. インスタンスメソッド

color#html()

色データを HTML で使われる "#rrggbb" の形式にした文字列を返します。

color#tolist():[alpha]

色データを赤・緑・青の順に並べたリストに変換します。アトリビュート :alpha をつけるとアルファ要素もいれ、赤・緑・青・アルファの順に並べたリストにします。

4.7.7. キャスト

以下のデータから color クラスのインスタンスにキャストできます。

- 色名を表す文字列またはシンボル
- 赤・緑・青または赤・緑・青・アルファ値を要素に持つリスト

4.8. dict クラス

4.8.1. インスタンスの生成

dict(elem[]?):[icase] {block?}

dict 型インスタンスを生成します。引数 elem にリスト形式で辞書データを指定します。
リストの内容はキーと値を以下のように並べたものになります。

```
dict([key, value, key, value, ..])
dict([[key, value], [key, value], ..])
dict([key => value, key => vakue, ..])
```

block を指定すると、その内容を辞書データに追加します。ブロックの内容はキーと値を以下のように並べたものになります。

```
dict {key, value, key, value, ..}
dict {[key, value], [key, value], ..}
dict {key => value, key => vakue, ..}
```

デフォルトでは、キーに文字列を指定した場合大文字と小文字を区別します。アトリビュート icase を指定すると、大文字・小文字を区別しない辞書を生成します。

`%{block}`

block の内容を辞書データに追加した dict 型インスタンスを生成します。ブロックの内容はキーと値を以下のように並べたものになります。

```
%{key, value, key, value, ..}
%{[key, value], [key, value], ..}
%{key => value, key => vakue, ..}
```

4.8.2. インスタンスメソッド

`dict#clear()`

辞書の内容を消去します。

`dict#erase(key):map`

引数 key で指定したキーに対応するエントリを削除します。

`dict#get(key, default?:nomap):map:[raise]`

引数 key で指定したキーに対応するエントリの値を返します。

対応するエントリが存在しない場合は default で指定した値を返します。default を省略したとき、この値は nil になります。

引数 default にはアトリビュート:nomap がついており、暗黙的マッピングの展開がされません。これにより、デフォルト値としてリストやイテレータを指定することができます。

アトリビュート :raise をつけると、対応するエントリが存在しない場合はエラーになります。default の値は無視されます。

`dict#gets(key, default?):map:[raise]`

引数 key で指定したキーに対応するエントリの値を返します。

対応するエントリが存在しない場合は default で指定した値を返します。default を省略したとき、この値は nil になります。

引数 default は暗黙的マッピングの対象になります。つまり、例えば key と default にリストが指定された場合、key[0]と default[0]、key[1]と default[1] … が対応するペアになります。

`dict#haskey(key):map`

引数 key で指定したキーに対応するエントリが存在するとき true、存在しない場合 false を返します。

`dict#items() {block?}`

キーと値を組にしたリストを順に返すイテレータを生成します。

block をつけると、データを抽出するごとにその内容を評価します。ブロックパラメータの形式は |item[], idx:number| で、item にキーと値を組にしたリスト、idx に 0 から始まるインデクス番号が

入ります。

`dict#keys() {block?}`

キーを順に返すイテレータを生成します。

`block` をつけると、データを抽出するごとにその内容を評価します。ブロックパラメータの形式は `|key, idx:number|` で、`key` にキー値、`idx` に 0 から始まるインデクス番号が入ります。

`dict#values() {block?}`

値を順に返すイテレータを生成します。

`block` をつけると、データを抽出するごとにその内容を評価します。ブロックパラメータの形式は `|value, idx:number|` で、`value` に値、`idx` に 0 から始まるインデクス番号が入ります。

`dict#len()`

辞書のサイズを返します。

`dict#set(key, value:nomap):map:reduce`

指定のキーと値を持ったエントリを追加します。`dict` インスタンス自身を戻り値として返します。

`dict#setdefault(key, value:nomap):map`

キーが存在しない場合、指定のキーと値を持ったエントリを追加して`value`の値を返しま。キーがすでに存在した場合は新たなエントリを追加せず、既存のエントリの値を返します。

`dict#sets(key, value):map:void`

`t.b.d.`

`dict#store(elems?):reduce:[default] {block?}`

引数 `elems` に指定したリストまたは `dict` 型の内容を追加します。

変数 `d` が `dict` のインスタンスとすると、リストの内容はキーと値を以下のように並べたものになります。

```
d.store([key, value, key, value, ..])
d.store([[key, value], [key, value], ..])
d.store([key => value, key => vakue, ..])
```

`block` を指定すると、その内容を辞書データに追加します。ブロックの内容はキーと値を以下のように並べたものになります。

```
d.store {key, value, key, value, ..}
d.store {[key, value], [key, value], ..}
d.store {key => value, key => vakue, ..}
```

アトリビュート `:default` をつけると、キーがすでに辞書に存在した場合何もしません。

4.9. `environment` クラス

4.9.1. インスタンスの生成

- 関数 `locals` でインスタンスを生成します。
- 関数 `outers` でインスタンスを生成します。

4.9.2. インスタンスメソッド

`environment#eval(expr:expr):map`

`environment` のスコープ内で `expr` の内容を評価します。

`environment#lookup(symbol:symbol, escalate:boolean => true):map`

`environment` 内で `symbol` に対応する定義値を返します。引数 `escalate` に `true` を指定すると、`environment` で定義値が見つからないとき外部スコープも探索します。

4.10. `error` クラス

4.10.1. インスタンスの生成

- 以下のインスタンスがあらかじめ定義されています。,
  `SyntaxError`, `Arithmetic`, `Error`, `TypeError`, `ZeroDivisionError`, `ValueError`, `SystemError`, `IOError`, `IndexError`, `KeyError`, `ImportError`, `AttributeError`, `StopIteration`, `RuntimeError`, `NameError`, `NotImplementedError`, `IteratorError`, `CodecError`, `CommandError`, `MemoryError`, `FormatError`, `ResourceError`
- 関数 `catch` のブロックパラメータとして渡されます。

4.10.2. インスタンスメソッド

t.b.d

4.11. `expr` クラス

4.11.1. インスタンスの生成

- Gura の任意の式の先頭にオペレータ "`" をつけると、`expr` クラスのインスタンスになります。

4.11.2. Expr 要素と判定メソッド

`expr` クラスは Gura 文法の構成要素である Expr 要素を表現します。Expr の要素と、それらのうちのどれを `expr` インスタンスが表現しているか判定するメソッドは以下のとおりです。

Expr 要素	判定メソッド
`Assign`	`expr#isassign()`
`Binary`	`expr#isbinary()`
`BinaryOp`	`expr#isbinaryop()`
`Block`	`expr#isblock()`
`BlockParam`	`expr#isblockparam()`
`Caller`	`expr#iscaller()`
`Container`	`expr#iscontainer()`
`DictAssign`	`expr#isdictassign()`
`Field`	`expr#isfield()`

Force	expr#isforce()
Indexer	expr#isindexer()
Lister	expr#islister()
Prefix	expr#isprefix()
Quote	expr#isquote()
String	expr#isstring()
Suffix	expr#issuffix()
Symbol	expr#issymbol()
Unary	expr#isunary()
UnaryOp	expr#isunaryop()
Value	expr#isvalue()

4.11.3. 要素を参照するインスタンスメソッド

`expr#block()`

caller 要素が持つ `block` の内容を `expr` 型で返します。

`expr#car()`

compound 要素が持つ `car` の内容を `expr` 型で返します。

`expr#cdr()`

compound 要素が持つ `cdr` の内容を `expr` 型で返します。

`expr#child()`

unary 要素が持つ `child` の内容を `expr` 型で返します。

`expr#each() {block?}`

container 要素が持つ子要素の内容を `expr` 型で返すイテレータを生成します。

`block` をつけると、子要素ごとにその内容を評価します。ブロックパラメータの形式は `|expr:expr, idx:number|` で、`expr` に子要素、`idx` に 0 から始まるインデクス番号が入ります。

`expr#getstring()`

string 要素の文字列データを返します。

`expr#getsymbol()`

symbol 要素のシンボル値を返します。

`expr#getvalue()`

value 要素の値を返します。

`expr#left()`

binary 要素の左側要素の内容を `expr` 型で返します。

`expr#right()`

binary 要素の右側要素の内容を expr 型で返します。

4.11.4. その他のインスタンスメソッド

`expr#eval()`

expr の内容を現在の環境で評価します。

`expr#exprname()`

要素名を文字列で返します。

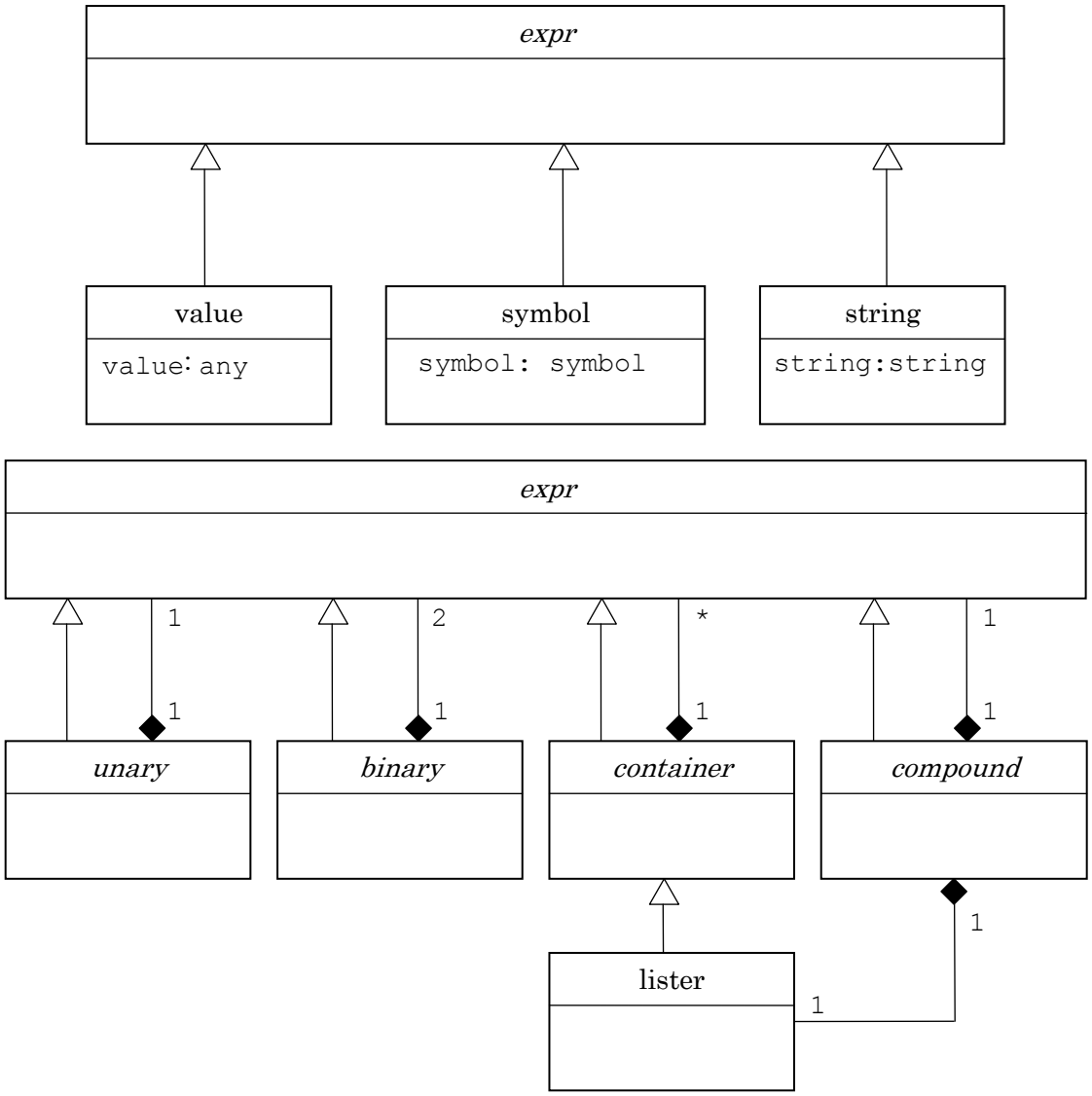
`expr#tofunction(`args*)`

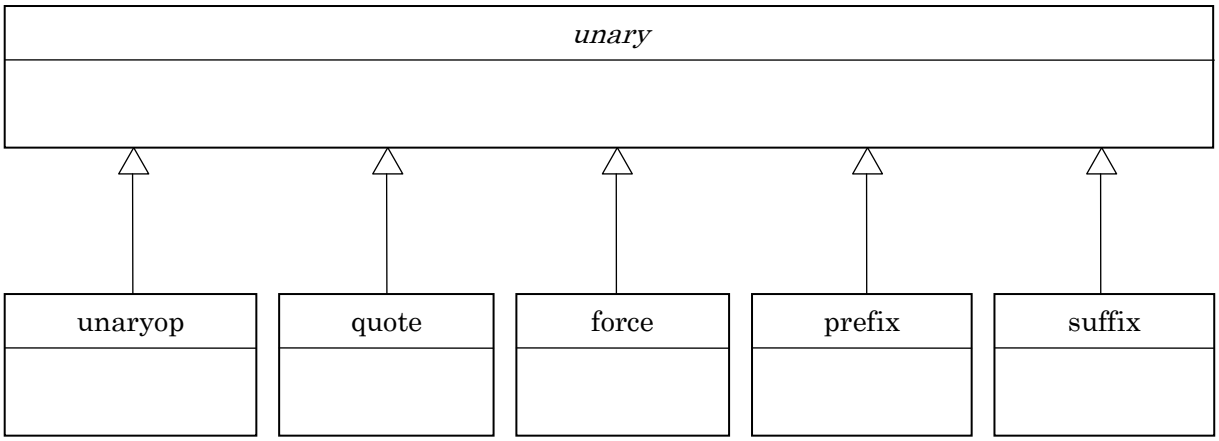
指定した引数列を持つ関数に変換します。

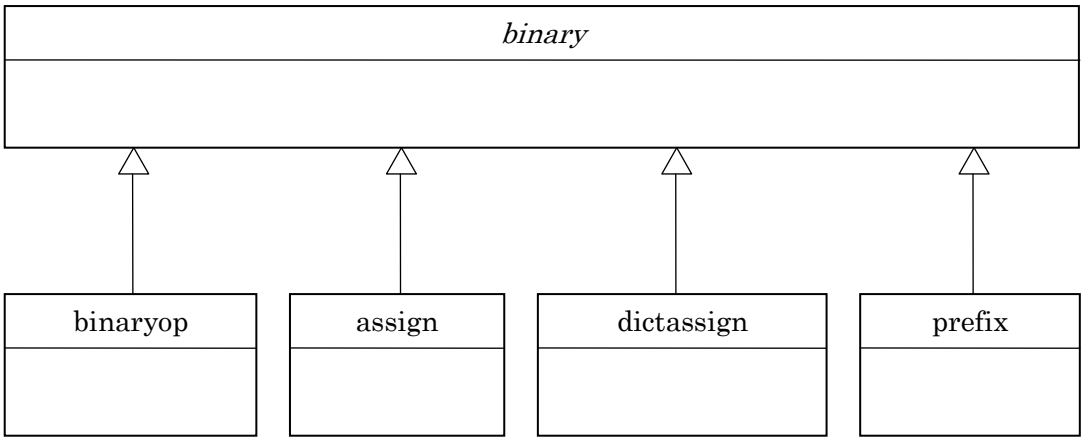
`expr#unquote()`

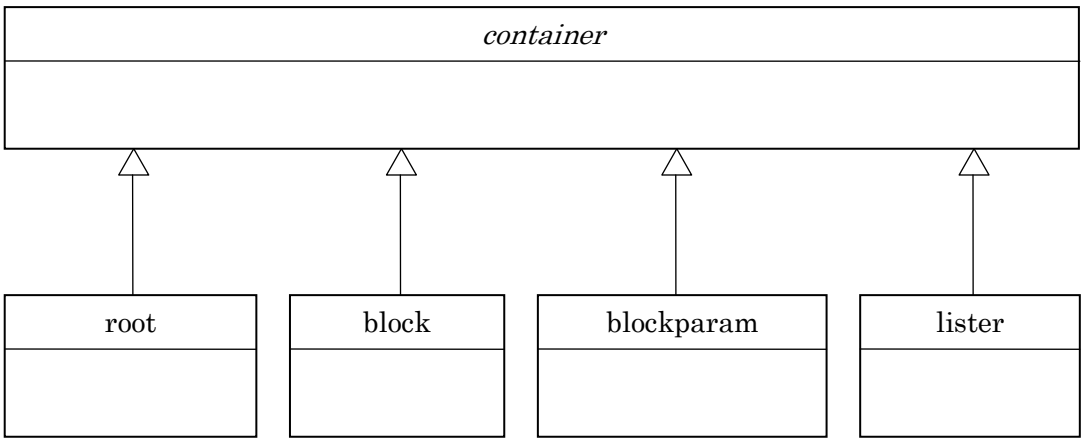
expr の内容が quote されているとき、それを取り除きます。

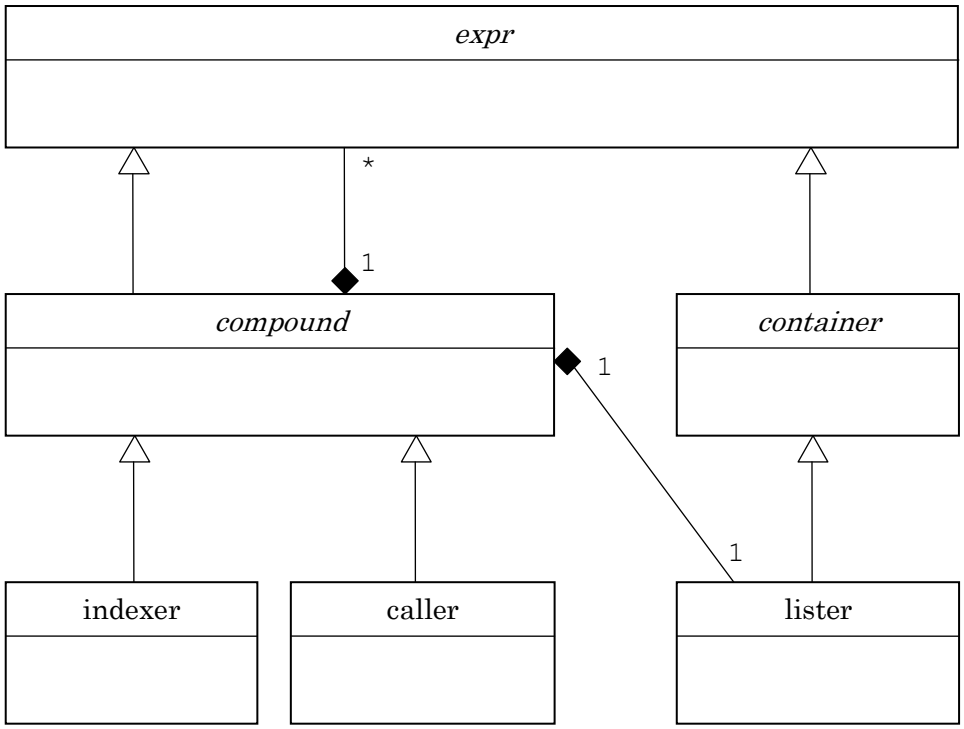
4.11.5. 式を構成する要素




unary
unaryop
quote
force
prefix
suffix


binary
binaryop
assign
dictassign
prefix


container
root
block
blockparam
lister



4.12. function クラス

4.12.1. インスタンスの生成

`function(`args*) {block}`

block に記述した手続きを持つ function 型インスタンスを生成して返します。引数リストを args で指定します。args が省略され、block にブロックパラメータがある場合、ブロックパラメータを引数リストとして扱います。

args もブロックパラメータも無い場合、block 中にドル記号 "$" を先頭に持つシンボルがあると、それらのシンボルを引数リストに追加します。引数のならびは出現した順になります。

`&{block}`

block に記述した手続きを持つ function 型インスタンスを生成して返します。block にブロックパラメータがある場合、ブロックパラメータを引数リストとして扱います。

ブロックパラメータが無い場合、block 中にドル記号 "$" を先頭に持つシンボルがあると、それらのシンボルを引数リストに追加します。引数のならびは出現した順になります。

4.12.2. インスタンスプロパティ

プロパティ	型	R/W	説明
symbol	symbol	R/W	関数のシンボルを返します
name	string	R/W	関数の名前を文字列で返します
fullname	string	R	関数のフルネームを文字列で返します。 ● インスタンスメソッドの場合: メソッドが属しているクラス名と関数名をシャープ記号 "#" でつなげた文字列を返します。

			● クラスメソッドの場合: メソッドが属しているクラス名と関数名をドット記号 "." でつなげた文字列を返します。 ● モジュール内の関数の場合: モジュール名と関数名をドット記号 "." でつなげた文字列を返します。
decls	iterator	R	引数宣言の情報を持った declaraion インスタンスを要素にするイテレータを返します。
expr	expr	R	関数の本体です
help	string	R/W	関数のヘルプを返します。 文字列を代入すると、それを新たなヘルプとして登録します。

4.12.3. インスタンスメソッド

function#diff(var?:symbol)

関数の内容を数学の微分公式に沿って微分し、その結果の式を function インスタンスで返します。引数 var で変数名のシンボルを指定すると、その変数に対する微分を行います。省略した場合、関数の最初の引数に対して微分を行います。

関数の内容は以下の条件を満たしている必要があります。

- 複数の式を含まないこと
- math モジュールの関数と四則演算およびべき乗からなる式であること

4.13. args クラス

4.13.1. インスタンスの参照

関数本体で__args__という名前の変数で内容を参照することができます。

4.13.2. インスタンスプロパティ

プロパティ	型	R/W	説明
values	any	R	引数に渡された値をリストにして返します

4.13.3. インスタンスメソッド

args#isset(symbol:symbol)

関数呼び出しで指定のシンボルのアトリビュートが指定されているか調べます。指定されていると true を返します。

args#quit_trailer():void

トレーラ関数が指定されていても、それを実行しないよう指示します。

4.14. declaration クラス

4.14.1. インスタンスの参照

function インスタンスの decls プロパティで参照することができます。

4.14.2. インスタンスプロパティ

プロパティ	型	R/W	説明
symbol	symbol	R	引数のシンボルを返します
name	string	R	引数の名前を文字列で返します
default	expr	R	デフォルト値の式を返します

4.15. image クラス

4.15.1. インスタンスの生成

image(stream:stream:r, format?:symbol, imgtype?:string):map {block?}

引数streamで指定したストリームを読み込んでイメージデータを構築します。引数formatは内部データ表現を表し、`rgb または`rgbaを指定します。読み込むデータのフォーマットは自動で識別されますが、引数 imgtype で明示的に指定することができます。指定可能なフォーマットは、モジュールをインポートすることで追加できます。

image(format:symbol):map {block?}

バッファを持たないイメージデータを作ります。引数 format は内部データ表現を表し、`rgb または `rgbaを指定します。

image(format:symbol, width:number, height:number, color?:color):map {block?}

指定のサイズを持ったブランクイメージデータを作ります。引数 formatは内部データ表現を表し、`rgbまたは`rgbaを指定します。widthおよびheightにそれぞれ幅と高さを指定します。デフォルトではイメージの内容は黒で塗りつぶされますが、引数 color で塗りつぶす色を指定できます。

4.16. インスタンスメソッド

image#allocbuff(width:number, height:number, color?:color):void

バッファを持たないイメージインスタンスに、指定の大きさのバッファを確保します。引数colorに、バッファを塗りつぶす色を指定します。省略した場合、黒で塗りつぶします。

image#crop(x:number, y:number, width?:number, height?:number):map

イメージの一部分をとりだして、新しい image インスタンスを生成します。引数 x, y に抽出する領域の左上座標を指定します。引数 width, height には、抽出する大きさを指定します。これらを省略した場合、イメージの右端および下端までを抽出します。

image#delpalette():reduce

イメージに関連付けられたパレットを削除します。

image#each(x?:number, y?:number, width?:number, height?:number, scandir?:symbol) {block?}

イメージのピクセル色データを順に走査して color 型のデータを返すイテレータを生成します。x、y、width、height に走査範囲、scandir に走査方向を指定します。scandir で指定できるシンボル値は以下のとおりです。

(scandir の指定はまだ未実装)。

block をつけると、ピクセルごとにその内容を評価します。ブロックパラメータの形式は |color:color, idx:number| で、color にピクセルの色データ、idx に 0 から始まるインデクス番号が入ります。

```
image#extract(x:number, y:number, width:number, height:number, element:symbol, dst):void
```

t.b.d.

```
image#fill(color:color):void
```

イメージ全体を指定した色で塗りつぶします。

```
image#fillrect(x:number, y:number,
                width:number, height:number, color:color):map:void
```

指定の範囲を指定した色で塗りつぶします。

```
image#flip(orient:symbol):map
```

イメージを左右または上下反転させた新しい image インスタンスを生成します。引数 orient に指定できるシンボル値は以下のとおりです。

- `horz　左右反転
- `vert　上下反転
- `both　左右および上下反転。これはイメージを 180 度回転させたことと同じです。

```
image#getpixel(x:number, y:number):map
```

指定の位置の色データを color 型で返します。

```
image#paste(x:number, y:number, src:image, width?:number, height?:number,
    xoffset:number => 0, yoffset:number => 0, alpha:number => 255):map:reduce
```

引数 x、y で指定した位置にイメージ src の画像内容をコピーします。引数 width、height はコピーする幅および高さを表し、これらを省略すると画像全体をコピーします。xoffset、yoffset はコピー元のオフセット座標です。引数 alpha を指定すると、コピーの際のブレンディング比率を指定できます。alpha が 0 で 0%、255 で 100%です。

```
image#putpixel(x:number, y:number, color:color):map:void
```

引数 x、y で指定した位置のピクセル色データを color に変更します。

```
image#read(stream:stream, imgtype?:string):map:reduce
```

引数 stream からイメージデータを読み込みます。このメソッドを実行するイメージインスタンスは、バッファが未確保である必要があります。すでにバッファを持っていた場合はエラーになります。

引数 imgtype には、"jpeg" や"png" というようにイメージタイプ名を文字列で指定します。この引数が省略されると、イメージファイルのヘッダ情報やファイル名のサフィックスからイメージタイプを識別します。

```
image#reducecolor(palette?:palette)
```

イメージデータ中の色データを、指定したパレット中の一番近いエントリの色で置き換えたイメージインスタンスを生成して返します。引数 palette を省略すると、イメージが持っているパレットを使って置き換えを行います。このとき、イメージにパレットがない場合はエラーになります。

image#replacecolor(colorOrg:color, color:color, tolerance?:number)

イメージ中 colorOrg と同じ色データを持つピクセルを color に置き換えます。引数 tolerance を指定すると、ピクセルごとに色データの値の差を算出し、それが tolerance 以下である場合に色データを置き換えます。

image#resize(width?:number, height?:number):map:[box]

イメージを指定の大きさにリサイズしたイメージインスタンスを生成して返します。

width および height にリサイズ結果の大きさを指定します。どちらかを省略した場合、オリジナルの縦横比率を保つようにリサイズされます。アトリビュート:box を指定して、width のみを指定すると、縦横がいずれも width の正方形が指定されます。

image#rotate(rotate:number, background?:color):map

イメージを引数 rotate で指定した角度だけ回転させたイメージインスタンスを生成して返します。

rotate の数値は degree で表わし、正の数が時計回り、負で反時計回りになります。

引数background は、回転させたときにできる余白を塗りつぶす色を指定します。省略すると、黒で塗りつぶします。

image#setalpha(alpha:number, color?:color, tolerance?:number):reduce

引数 color で指定した色データを持つピクセルのアルファ値を引数 alpha の値に置き換えます。引数 color を省略すると、イメージ全体のアルファ値を alpha の値にします。引数 tolerance を指定すると、ピクセルごとに色データの値の差を算出し、それが tolerance 以下である場合にアルファ値を置き換えます。

image#size()

イメージの幅と高さをリストにして返します。

image#store(x:number, y:number, width:number, height:number, element:symbol, src):void

t.b.d.

image#thumbnail(width?:number, height?:number):map:[box]

イメージデータを、縦横比を保存しながら幅 width、高さ height の範囲内に収まるようリサイズしたイメージインスタンスを生成して返します。指定した範囲よりもイメージが小さい場合、元のイメージへの参照をそのまま返します。引数 width のみを指定してアトリビュート :box をつけると、幅・高さとも width ピクセルの範囲に収まるイメージを生成します。

image#write(stream:stream, imgtype?:string):map:reduce

引数 stream にイメージデータを書き込みます。このメソッドを実行するイメージインスタンスは、バッファを持っている必要があります。バッファを持っていない場合はエラーになります。

引数 imgtype には、"jpeg" や"png" というようにイメージタイプ名を文字列で指定します。この引数が省略されると、ストリームについているファイル名のサフィックスからイメージタイプを識別します。

4.17. iterator クラス

4.17.1. インスタンスの生成

`iterator(value+) {block?}`

汎用イテレータ関数です。指定された要素を順次返すイテレータを生成します。

要素がイテレータやリストの場合、それらの要素を返していきます。

`consts(value, num?:number) {block?}`

指定の値を指定の数だけ出力するイテレータを生成します。

引数 value に値、num に生成する個数を指定します。value には任意の型のデータを指定できます。

num を省略すると、無限に値を返すイテレータになります。

`range(num:number, num_end?:number, step?:number):map {block?}`

開始値と終了値、および間隔を指定して連続する数列を出力するイテレータを生成します。

引数 num のみを指定すると、0 から num - 1 までの整数を出力します。

引数 num と num_end を指定すると、num から num_end - 1 までの整数を出力します。

引数 num, num_end および step を指定すると、num を開始値にして、step ごとに数値をインクリメントして num_end を超えない範囲までの数値を出力します。

連続した数値を出力するのに、オペレータ ".." を使うこともできます。"n..m" という形式では、n から m までの整数を出力するイテレータになります。また、"n.." と指定すると、n を始点にして、無限にインクリメントするイテレータになります。

`interval(a:number, b:number, samples:number):map:[open,open_l,open_r] {block?}`

範囲とサンプル数を指定して数列を出力するイテレータを生成します。一般式は以下のとおりです。 引数 a に最小値、b に最大値を指定すると、サンプル数 samples 個だけ、[a, b] の範囲内で等間隔な数列を出力します。

アトリビュート:open, :open_l, :open_r を指定すると、範囲のオープン条件を指定できます。:open を指定すると、範囲指定が(a, b)に、:open_l では(a, b]、:open_r では[a, b)になります。

4.17.2. インスタンスメソッド

イテレータが実装するメソッドは、リストのメソッドと大部分が共通しています。共通しているメソッドは list クラスの項に掲載していますので、そちらを参照ください。この項は、イテレータ特有のメソッドを示します。

`iterator#delay(delay:number) {block?}`

イテレータの要素を返すたびに引数 delay で指定した秒数だけ遅延します。

block をつけると、イテレータの要素ごとにその内容を評価します。ブロックパラメータの形式は |value, idx:number| で、value に要素データ、idx に 0 から始まるインデクス番号が入ります。

`iterator#isinfinite()`

イテレータが無限イテレータのとき true、有限イテレータのとき false を返します。

`iterator#next()`

イテレータの次の要素の値を返します。

`iterator#repeater()`

イテレータをリピータとして設定します。

4.18. `list` クラス

4.18.1. インスタンスの生成

`list(iter+:iterator), xlist(iter+:iterator)`

ひとつ以上のイテレータを結合した結果を、ひとつのリストインスタンスとして返します。`xlist` は、要素から `nil` 値を取り除きます。

`@(func?:function) {block?}`

ブロックの要素をもとにリストを生成します。詳細は「Gura 言語マニュアル」を参照ください。

`dim(n+:number) {block?}`

指定の要素数を持った多重リストを生成して返します。例えば、`dim(2, 3)` は `[[nil, nil, nil], [nil, nil, nil]]` というリストを生成します。要素の値はデフォルトで `nil` ですが、`block` を指定するとブロックの評価値を要素の値とします。ブロックの評価の際`|i0:number, i1:number, ..|` という形式のブロック引数を渡します。`i0, i1 …` はループインデクスです。

`set(iter+:iterator):[and,or,xor], xset(iter+:iterator):[and,or,xor]`

ひとつ以上のイテレータを結合し、重複した要素をとりのぞいた結果をひとつのリストインスタンスとして返します。`xset` は、要素から `nil` 値を取り除きます。デフォルトでは、イテレータ同士 `or` 論理で結合します。アトリビュート `and` を指定すると、イテレータ間で同じ値を持つ要素のみを抽出します。アトリビュート `xor` を指定すると、イテレータ間で重複しない要素のみを抽出します。

4.18.2. インスタンスメソッド

`list#add(elem+):reduce`

リストに、引数 `elem` で表わされる要素を追加します。これは破壊的メソッドです。

`list#align(n:number, value?):map {block?} / iterator#align(n:number, value?) {block?}`

リストやイテレータの要素中、`n` 個までの要素を返すイテレータを生成します。リストの要素数が引数 `n` よりも小さい場合、実際の要素数を超えた分は `value` の値を返します。`value` が省略されたとき、その部分は `nil` になります。

`block` をつけると、イテレータの要素ごとにその内容を評価します。ブロックパラメータの形式は `|value, idx:number|` で、`value` に要素データ、`idx` に 0 から始まるインデクス番号が入ります。

`list#and() / iterator#and()`

要素間をオペレータ `"&"` で演算した結果を返します。

`list#append(elem+):reduce`

リストに、引数 `elem` で表わされる要素を追加します。これは破壊的メソッドです。`elem` がリストまたはイテレ

ータのとき、それらの要素が追加対象になります。

`list#average() / iterator#average()`

要素から平均値を算出し、結果を返します。

`list#clear():reduce`

要素をすべてとりのぞき、空のリストにします。これは破壊的メソッドです。

`list#combination(n:number) {block?}`

リストから、重複しない n 個のデータの組み合わせをリストにして返すイテレータを生成します。イテレータの要素数は、リストのデータ数を m 個としたとき ${}_mC_n$ になります。

block をつけると、組み合わせのリストごとにその内容を評価します。ブロックパラメータの形式は |elements:list, idx:number| で、elements に組み合わせリスト、idx に 0 から始まるインデクス番号が入ります。

`list#count(criteria?) / iterator#count(criteria?)`

リストまたはイテレータ中、条件に合致する要素の数を返します。条件 criteria には値または関数を指定します。

criteria を省略すると、要素中で真値と判断できるものの数を返します。

criteria に値を指定した場合、その値と要素を比較し、等しいと判断したものの数を数えます。

criteria に渡す関数は、引数を一つとり boolean 値を返すものを指定します。list#count メソッドは要素をひとつずつ関数に渡し、帰ってきた true の数を数えます。

`list#each() {block?} / iterator#each() {block?}`

リストまたはイテレータの要素を順に走査するイテレータを返します。

block をつけると、要素ごとにその内容を評価します。ブロックパラメータの形式は |value, idx:number| で、value に要素値、idx に 0 から始まるインデクス番号が入ります。

`list#erase(idx*:number):reduce`

引数 idx で指定される位置の要素をリストから削除します。これは破壊的メソッドです。

`list#filter(criteria) {block?} / iterator#filter(criteria) {block?}`

リストまたはイテレータ中、引数 criteria で指定した条件に合致する要素を返すイテレータを生成します。

criteria に関数を指定すると、各要素を引数にしてその関数を呼び出し、関数が true を返したときの要素を抽出します。

criteria にリストまたはイテレータを指定すると、criteria 中で true となる位置のデータを抽出します。

block をつけると、要素ごとにその内容を評価します。ブロックパラメータの形式は |value, idx:number| で、value に要素値、idx に 0 から始まるインデクス番号が入ります。

`list#find(criteria?):[index] / iterator#find(criteria?):[index]`

リストまたはイテレータ中、引数 criteria で指定した条件に合致する要素の値を返します。criteria を省略すると、true と判定される要素の値を返します。アトリビュートに:index を指定すると、要素の値

ではなくインデクス値を返します。

criteria に指定した値の型によって、以下の判定処理を行います。

- 関数を指定すると、criteria(x) という形式で要素をその関数の引数として渡し、戻り値が true か否かをチェックします。
- リストまたはイテレータを指定すると、その要素が true か否かをチェックします。
- その他の値を指定すると、要素がその値と等しいかチェックします。

list#first()

リストの最初の要素値を返します。

list#flat()

入れ子になったリストをすべて一次元に展開したリストを返します。

list#fold(n:number, nstep?:number):[iteritem] {block?} / iterator#fold(n:number):[iteritem] {block?}

リストまたはイテレータの要素から n 個ずつ組にしたリストを返すイテレータを生成します。引数 nstep を指定すると、次に抽出する要素の間隔を指定できます。nstep を省略すると、次の抽出位置は n 個先になります。アトリビュート:iteritem をつけると、組にした結果をリストではなくイテレータで返します。

block をつけると、抽出したリストごとにその内容を評価します。ブロックパラメータの形式は |elements:list, idx:number| で、elements に抽出リスト、idx に 0 から始まるインデクス番号が入ります。

list#format(format:string):map / iterator#format(format:string) {block?}

printf 関数のフォーマットで、リストまたはイテレータの要素を文字列に変換します。

list#get(index:number):map:flat

リスト中、引数 index で指定した位置にあるデータを取得します。

list#head(n:number):map {block?} / iterator#head(n:number):map {block?}

リストまたはイテレータの最初の n 個のデータを返すイテレータを生成します。

block をつけると、データごとにその内容を評価します。ブロックパラメータの形式は |value, idx:number| で、value にデータ値、idx に 0 から始まるインデクス番号が入ります。

list#iscontain(value) / iterator#iscontain(value)

リストまたはイテレータの要素中に、value と同じ値のデータがある場合 true を、ない場合は false を返します。

list#isempty()

リストの要素が空のとき true を返します。ひとつでも要素があれば false を返します。

list#join(sep:string => "") / iterator#join(sep?:string)

リストまたはイテレータの要素を文字列に変換し、それらを指定の文字列 sep で連結します。

iterator#joinb()

イテレータで返される binary 型の要素を連結した結果を返します。要素が binary 型でない場合、エラ

ーになります。

`list#last()`

リストの最後の要素を返します。

`list#len() / iterator#len()`

リストまたはイテレータの要素の数を返します。

`list#map(func:function) {block?} / iterator#map(func:function) {block?}`

リストまたはイテレータから要素の値を関数オブジェクト `func` に引数として渡した結果を返すイテレータを生成します。関数の呼び出しは、要素の値を`value`とすると`func(value)` という形式になります。ブロックを指定すると、生成したイテレータを即座に評価します。このとき、ブロックパラメータの形式は `|valueMapped, idx:number|` となり、`valueMapped` に関数 `func` の戻り値、`idx` に 0 から始まるインデクス番号が入ります。

`list#max():[index,last_index,indices] / iterator#max():[index,last_index,indices]`

アトリビュートを何もつけずに実行したとき、リストまたはイテレータの要素のうち、大小比較をした結果が最も大きかった値を返します。

アトリビュート`:index` をつけると、最も大きな値が最初に見つかったインデクスを返します。アトリビュート`:last_index` では、最も大きな値が最後に見つかったインデクスを返します。

アトリビュート`:indices` をつけると、最も大きな値が複数あった場合、それらすべてのインデクス値をリストにして返します。

`list#min():[index,last_index,indices] / iterator#min():[index,last_index,indices]`

アトリビュートを何もつけずに実行したとき、リストまたはイテレータの要素のうち、大小比較をした結果が最も小さかった値を返します。

アトリビュート`:index` をつけると、最も小さな値が最初に見つかったインデクスを返します。アトリビュート`:last_index` では、最も小さな値が最後に見つかったインデクスを返します。

アトリビュート`:indices` をつけると、最も小さな値が複数あった場合、それらすべてのインデクス値をリストにして返します。

`list#nilto(replace) / iterator#nilto(replace)`

リストまたはイテレータの要素が `nil` のとき、指定した値に変換します。

`list#offset(n:number):map {block?} / iterator#offset(n:number) {block?}`

リストまたはイテレータの先頭から指定の数だけ除外した後の要素を返すイテレータを生成します。

`list#or() / iterator#or()`

要素間をオペレータ `"|"` で演算した結果を返します。

`list#pack(format:string) / iterator#pack(format:string) {block?}`

引数 `fomat` で指定したフォーマットに基づいて、リストまたはイテレータの要素を埋め込んだバイナリデータを `binary` 型として返します。フォーマットの詳細は `pack` 関数の説明を参照してください。

`list#permutation(n?:number) {block?}`

リストから、重複しない n 個のデータの順列組み合わせをリストにして返すイテレータを生成します。イテレータの要素数は、リストのデータ数を m 個としたとき ${}_mP_n$ になります。

block をつけると、順列組み合わせのリストごとにその内容を評価します。ブロックパラメータの形式は |elements:list, idx:number| で、elements に順列組み合わせリスト、idx に 0 から始まるインデクス番号が入ります。

`list#pingpong(n?:number):[sticky,sticky_l,sticky_r] {block?} /`
`iterator#pingpong(n?:number):[sticky,sticky_l,sticky_r] {block?}`

リストまたはイテレータの要素を順に走査し、最後に到達したら逆向きに走査、再び最初に戻ったら順に走査を繰り返すイテレータを生成します。

引数 n に走査結果で得られる要素の数を指定します。この引数を省略すると、無限に走査をくりかえす無限イテレータになります。アトリビュート:sticky, :sticky_l, :sticky_r は先頭または終端で折り返しをするときに要素を 2 度繰り返すか否かを指定します。:sticky_l が先頭要素、:sticky_r が終端要素、:sticky が両端の要素に対する繰り返し指定になります。

`iterator#print(stream?:stream:w)`

要素の値を文字列にして stream に出力します。stream を省略した場合、標準出力に出力します。

`list#printf(format:string, stream?:stream:w):void /`
`iterator#printf(format:string, stream?:stream:w)`

要素の値を printf のフォーマットに従って文字列にし、stream に出力します。stream を省略した場合、標準出力に出力します。

`iterator#println(stream?:stream:w)`

要素の値と改行を stream に出力します。

`list#rank(directive?):[stable] / iterator#rank(directive?) {block?}`

要素の順番を並べ替えたとき、各要素が位置するインデクス番号を返すイテレータを生成します。

並べ替えの順序はデフォルトで昇順ですが、引数 directive にシンボルまたは関数を指定することで順序を指示することができます。directive に、シンボル`ascend を指定すると昇順、シンボル`descend を指定すると降順になります。

directive に関数を渡す場合、この関数は二つの引数をとり、-1, 0, +1 のいずれかの整数値を返すものである必要があります。今、関数の一般式が f(a, b) であるとすると、以下のような値を返すようにします。

昇順: a < b のとき -1、 a == b のとき 0、 a > b のとき +1
降順: a > b のとき +1、 a == b のとき 0、 a < b のとき +1

アトリビュート:stable をつけると、ステーブルソートになります。大小比較が等しい要素が複数あったとき、それらの順序がソート前と同じである保障が得られます。

`list#reduce(accum) {block} / iterator#reduce(accum) {block}`

要素に対し畳み込み操作を行います。関数を実行すると、リストまたはイテレータから要素をひとつ受取り、

この値 value と累積結果 accum の値を|value, accum| というブロックパラメータの形式でブロックに渡します。ブロックの評価結果を新たな accum とし、以下同じ操作を要素ごとに繰り返します。

```
list#replace(value, replace) / iterator#replace(value, replace)
```

要素が value に等しいとき、replace に置き換えるイテレータを生成します。

```
list#reverse() {block?} / iterator#reverse() {block?}
```

要素列を逆から走査するイテレータを生成します。

```
list#round(n?:number) {block?} / iterator#round(n?:number) {block?}
```

リストまたはイテレータの要素を順に走査し、最後に到達したら再び最初に戻るイテレータを生成します。
引数 n に走査結果で得られる要素の数を指定します。この引数を省略すると、無限に走査をくりかえす無限イテレータになります。

```
list#runlength() {block?} / iterator#runlength() {block?}
```

リストまたはイテレータの要素を順に走査し、連続した数とその値をペアにしたものを要素に返すイテレータを生成します。

```
list#shift():[raise]
```

リストから最初の要素をとりのぞき、その値を返します。
リストが空の時、デフォルトでは何もせず nil を返します。アトリビュート:raise をつけると空のリストにたいしてこのメソッドを実行するとエラーを発生させます。

```
list#shuffle():reduce
```

リスト要素の順番を乱数で入れ替えた結果をリストにして返します。

```
list#since(criteria) {block?} / iterator#since(criteria) {block?}
```

リストまたはイテレータから、条件に合致した時点からの要素を抽出するイテレータを生成します。
criteria には関数またはイテレータを指定できます。
関数は、一つの引数をとり boolean 値を返すものを指定します。since 関数はリストまたはイテレータの要素をひとつずつ関数に渡し、その戻り値が true になった時点で抽出を開始します。
criteria にイテレータを指定すると、since 関数は抽出対象のリストまたはイテレータと同時に criteria のイテレータを走査し、これが true 値になった時点で抽出を開始します。

```
list#skip(n:number):map {block?} / iterator#skip(n:number) {block?}
```

指定数だけ要素を除外しながら要素列を走査するイテレータを返します。引数 n に除外する要素数を指定します。

```
list#skipnil() {block?} / iterator#skipnil() {block?}
```

nil 要素をとりのぞくイテレータを生成します。

```
list#sort(directive?, keys[]?):[stable] {block?} /
                    iterator#sort(directive?, keys[]?):[stable] {block?}
```

要素の順番を並べ替えた結果をイテレータで返します。リストで結果を得る場合はアトリビュート:list を指

定します。

並べ替えの順序はデフォルトで昇順ですが、引数 directive にシンボルまたは関数を指定することで順序を指示することができます。directive に、シンボル`ascend を指定すると昇順、シンボル`descend を指定すると降順になります。

directive に関数を渡す場合、この関数は二つの引数をとり、-1, 0, +1 のいずれかの整数値を返すものである必要があります。今、関数の一般式が f(a, b) であるとすると、以下のような値を返すようにします。

昇順: a < b のとき -1、 a == b のとき 0、 a > b のとき +1

降順: a > b のとき +1、 a == b のとき 0、 a < b のとき +1

sort メソッドは、デフォルトではリストの要素そのものの大小で並び替えを行いますが、引数 keys にリストを渡すと、これをキーとしてソート処理をします。keys の要素数はリストの要素数と同じでなければいけません。

アトリビュート:stable をつけると、ステーブルソートになります。大小比較が等しい要素が複数あったとき、それらの順序がソート前と同じである保障が得られます。

list#stddev() / iterator#stddev()

要素から標準偏差を算出し、結果を返します。

list#sum() / iterator#sum()

すべての要素を加算した結果を返します。

list#tail(n:number):map {block?} / iterator#tail(n:number) {block?}

リストまたはイテレータの最後から指定の数の要素だけ返すイテレータを生成します。

list#variance() / iterator#variance()

要素から分散値を算出し、結果を返します。

list#while (criteria) {block?}

リストまたはイテレータから、条件に合致している間の要素を抽出するイテレータを生成します。

criteria には関数またはイテレータを指定できます。

関数は、一つの引数をとり boolean 値を返すものを指定します。while 関数はリストまたはイテレータの要素をひとつずつ関数に渡し、その戻り値が true の間だけ要素を抽出します。false になったら処理を終了します。

criteria にイテレータを指定すると、while 関数は抽出対象のリストまたはイテレータと同時に criteria のイテレータを走査し、これが true 値の間だけ要素を抽出します。false になったら処理を終了します。

4.19. matrix クラス

4.19.1. インスタンスの生成

matrix(nrows:number, ncols:number, value?)

指定のサイズをもつ matirx 型インスタンスを生成します。引数 nrows に行数、ncols に桁数を指定し

ます。引数 value に要素の値を指定します。value を省略すると、要素の値は nil になります。

@@{block}

block の内容を要素にした matrix インスタンスを生成します。ブロックの内容は値を以下のように並べたものになります。

スクリプト一般式	生成されるマトリクス
@@{{a11, a12, a13, …}, {a21, a22, a23, …}, …}	$\begin{pmatrix} a11 & a12 & a13 & \cdots \\ a21 & a22 & a23 & \cdots \\ & \vdots & & \end{pmatrix}$
@@{a11, a21, a31, …}	$\begin{pmatrix} a11 \\ a21 \\ \vdots \end{pmatrix}$
@@{{a11, a12, a13, …}}	$\begin{pmatrix} a11 & a12 & a13 & \cdots \end{pmatrix}$

4.19.2. インデクスによる要素操作

数学表記では行と列のインデクスは 1 から始まりますが、matrix インスタンスにおけるインデクスの開始は 0 になります。

matrix インスタンスを m としたとき、row 行 col 列の要素は m[row][col]と表すことができます。

4.19.3. クラスメソッド

matrix.identity(n:number):static:map {block?}

指定の大きさの単位行列を生成します。

matrix.rotation(angle:number, tx?:number, ty?:number)
:static:map:[deg] {block?}

平面に対する回転行列を返します。回転の方向は反時計まわりです。引数 angle の単位はラジアンですが、アトリビュート:deg をつけると degree 値で指定することができます。引数 tx, ty を指定すると、平行移動の成分を含めた行列を返します。

matrix.rotation_x(angle:number, tx?:number, ty?:number, tz?:number)
:static:map:[deg] {block?}

三次元空間で、x 軸を中心にした回転行列を返します。回転の方向は y 軸を z 軸に向ける方向です。引数 angle の単位はラジアンですが、アトリビュート:deg をつけると degree 値で指定することができます。引数 tx, ty, tz を指定すると、平行移動の成分を含めた行列を返します。

matrix.rotation_y(angle:number, tx?:number, ty?:number, tz?:number)
:static:map:[deg] {block?}

三次元空間で、y 軸を中心にした回転行列を返します。回転の方向は z 軸を x 軸に向ける方向です。引数

angle の単位はラジアンですが、アトリビュート:deg をつけると degree 値で指定することができます。引数 tx, ty, tz を指定すると、平行移動の成分を含めた行列を返します。

```
matrix.rotation_z(angle:number, tx?:number, ty?:number, tz?:number)
                                                    :static:map:[deg] {block?}
```

三次元空間で、z 軸を中心にした回転行列を返します。回転の方向は x 軸を y 軸に向ける方向です。引数 angle の単位はラジアンですが、アトリビュート:deg をつけると degree 値で指定することができます。引数 tx, ty, tz を指定すると、平行移動の成分を含めた行列を返します。

4.19.4. インスタンスメソッド

```
matrix#col(col:number):map
```

指定の列の要素をリストにして返します。

```
matrix#colsize()
```

マトリクスの列の数を返します。

```
matrix#each():[transpose]
```

マトリクスの要素をひとつずつとりだすイテレータを生成します。

デフォルトでは、先頭の行から順に左から右へ横方向に要素をとりだします。つまり、2 行 3 列のマトリクス m があったとき、each メソッドが返す要素は m_{11}, m_{12}, m_{13}, m_{21}, m_{22}, m_{23} となります。

アトリビュート :transpose をつけると、左端の列から順に上から下へ縦方向に要素を取り出します。同じく 2 行 3 列のマトリクス m を考えると、each メソッドが返す要素は m_{11}, m_{21}, m_{12}, m_{22}, m_{13}, m_{23} となります。

```
matrix#eachcol()
```

列要素をリストにして返すイテレータを生成します。

```
matrix#eachrow()
```

行要素をリストにして返すイテレータを生成します。

```
matrix#invert()
```

逆行列を計算し、結果を返します。

```
matrix#issquare()
```

正方行列のとき true、それ以外は false を返します。

```
matrix#row(row:number):map
```

指定の行の要素をリストにして返します。

```
matrix#rowsize()
```

マトリクスの行の数を返します。

```
matrix#set(value)
```

すべての要素を value で置き換えます。

matrix#setcol(col:number, value)

指定の列の要素を value で置き換えます。

matrix#setrow(row:number, value)

指定の行の要素を value で置き換えます。

matrix#submat(row:number, col:number, nrows:number, ncols:number):map

部分行列を返します。

matrix#tolist():[transpose]

マトリクスを二次元のリストにして返します。アトリビュート:transpose をつけると、転置した結果を二次元リストに変換します。

matrix#transpose()

転置行列を返します。

matrix#roundoff(threshold:number => 1e-10)

指定した値ですべての要素を丸めます。

4.20. palette クラス

4.20.1. インスタンスの生成

palette(type)

palette 型インスタンスを生成します。

引数 type に数値を指定すると、その数だけのエントリを持ったパレットを生成します。

引数 type に以下のいずれかのシンボルを指定し、既存のエントリを持ったパレットを生成することもできます。

`basic　16 個の基本色エントリを持つパレット

`win256　Windows の 256 色パレット

`websafe　Web-safe な 216 色を持つパレット。インデクス 216 から 255 までは黒になります。

4.20.2. インスタンスメソッド

palette#each() {block?}

パレット中の色データを color 型で返すイテレータを生成します。

block をつけると、色データごとにその内容を評価します。ブロックパラメータの形式は |color:color, idx:number| で、color に色データ、idx に 0 から始まるインデクス番号が入ります。

palette#nearest(color:color):map:[index]

パレットのエントリ中、引数で指定した色に最も近い色データを返します。アトリビュート :index をつけると、エントリ中のインデクス番号を返します。

palette#shrink():reduce:[align]

パレットの未使用エントリを削除して、エントリの格納に必要なサイズに変更します。アトリビュート :align

を指定すると、エントリの格納に必要な最小の 2 のべき乗のサイズにします。

palette#updateby(image_or_palette):reduce:[shrink,align]

イメージまたは他のパレットでから取得した色データで、パレットのエントリを更新します。パレットのエントリサイズを超える数の色データがあった場合、超えた分は無視されます。アトリビュート :shrink をつけると、更新後に未使用エントリを削除して、エントリの格納に必要なサイズに変更します。このとき、アトリビュート :align も指定すると、エントリの格納に必要な最小の 2 のべき乗のサイズにします。

4.21. semaphore クラス

4.21.1. インスタンスの生成

semaphore()

semaphore 型インスタンスを生成します。

4.21.2. インスタンスメソッド

semaphore#wait()

セマフォが解放されるのを待ち、解放されたら所有権を取得します。semaphore#release と対にして使用します。

semaphore#release()

セマフォの所有権を解放します。semaphore#wait と対にして使用します。

semaphore#session() {block}

セマフォの所有権を取得してblockを評価し、評価後に所有権を解放します。これはsemaphore#waitとsemaphore#releaseをブロックの入り口と出口で実行したのと同じです。

4.22. stream クラス

4.22.1. インスタンスの生成

stream(name:string, mode?:string, codec?:codec):map {block?}
open(name:string, mode?:string, codec?:codec):map {block?}

引数 name で指定したパス名のストリームをオープンします。引数 mode にはストリームのアクセス方法を以下の文字で指定します。

r	読み込みモード
w	書き込みモード
a	追加書き込みモード

引数 mode を省略し、パス名の先頭に ">" をつけると、書き込みモードでストリームをオープンします。
引数 codec は、ストリームを文字列として読み書きするときのコーデックを指定します。この指定は、ストリームをバイナリデータとして扱うメソッドや関数には影響しません。

4.22.2. ストリーム操作を行うグローバル関数

`copy(src:stream:r, dst:stream:w, bytesunit:number => 65536):map:void {block?}`

入力用ストリーム `src` から出力用ストリーム `dst` にデータをコピーします。コピー処理はデフォルトで 65536 バイト単位で行われますが、引数 `bytesunit` でこれを変更することができます。

`block` を指定すると、コピー単位ごとにブロックパラメータを `|buff:binary|` という形式でブロックに渡して評価します。`buff` は入力用ストリームから入力したデータが入り、評価結果が `binary` 型の値であればそれを出力用ストリームに出力します。それ以外の型の場合はもとのデータを出力します。

`template(src:stream:r, dst?:stream:w):map:[lasteol,noindent] {block?}`

入力用ストリーム `src` からテンプレート文字列を読み込み、スクリプトの評価結果を埋め込んだ文字列を出力用ストリーム `dst` に出力します。引数 `dst` を省略すると、結果は `string` 型のデータで戻り値として返されます。

スクリプトはテキスト中に `"${"` と `"}"` に囲まれた領域に記述されます。

スクリプトの評価結果で、最後に現れた改行コードはとりのぞかれます。アトリビュート `:lasteol` をつけると、この改行コードをとりのぞかずに出力に含めます。

スクリプトの出力結果が複数行にわたるとき、スクリプトの開始を表す `"${"` の行の先頭にある空白文字が各行に追加されます。アトリビュート `:noindent` をつけると、このインデンテーション機能を無効にします。

`readlines(stream?:stream:r):[chop] {block?}`

入力用ストリーム `stream` からテキストを読み込み、行ごとに分割した文字列を返すイテレータを生成します。デフォルトでは、改行記号まで含めた文字列を返しますが、アトリビュート `:chop` をつけると、改行記号を削除します。

引数 `stream` を省略すると、標準入力からテキストを読み込みます。

`block` をつけると、行ごとにその内容を評価します。ブロックパラメータの形式は `|line:string, idx:number|` で、`line` に行ごとの文字列、`idx` に 0 から始まるインデクス番号が入ります。

4.22.3. インスタンスメソッド

`stream#close()`

ストリームをクローズします。

ストリームインスタンスは消滅するときに自動的にクローズ処理を行いますが、`close` メソッドをそれを明示的に行います。

`stream#compare(stream:stream:r):map`

ストリームの内容をバイトごとに比較します。要素の数もデータ内容も同じ場合 `true` を返します。それ以外は `false` を返します。

`stream#delcr(flag?:boolean):reduce`

引数なしで実行するか、`flag` に `true` を指定すると、テキストデータを読み込む際に改行コードを CR-LF から LF に変換します。

`stream#addcr(flag?:boolean):reduce`

引数なしで実行するか、`flag` に `true` を指定すると、テキストデータを出力する際に改行コードを LF から CR-LF に変換します。

`stream#flush():void`

ストリームへの出力内容をフラッシュします。

`stream#peek(len?:number)`

ストリームからデータを `len` バイトだけ先読みします。シーク位置は変化しません。

`stream#print(values*):map:void`

引数に指定した値の内容を文字列にしてストリームに出力します。

`stream#printf(format:string, values*):map:void`

`printf` のフォーマットに基づいてストリームに出力します。

`stream#println(values*):map:void`

引数に指定した値の内容を文字列にしてストリームに出力します。最後に改行コードをストリームに出力します。

`stream#read(len?:number)`

ストリームからデータを `len` バイトだけ読み込みます。シーク位置も `len` だけ先に進みます。

`stream#readline():[chop]`

入力用ストリーム `stream` からテキストを一行分読み込み、文字列にして返します。デフォルトでは、改行記号まで含めた文字列を返しますが、アトリビュート `:chop` をつけると、改行記号を削除します。ファイルの末尾に到達した場合は `nil` を返します。

`stream#readlines(nlines?:number):[chop] {block?}`

入力用ストリーム `stream` からテキストを読み込み、行ごとに分割した文字列を返すイテレータを生成します。デフォルトでは、改行記号まで含めた文字列を返しますが、アトリビュート `:chop` をつけると、改行記号を削除します。

`block` をつけると、行ごとにその内容を評価します。ブロックパラメータの形式は `|line:string, idx:number|` で、`line` に行ごとの文字列、`idx` に 0 から始まるインデクス番号が入ります。

`stream#readtext()`

ストリームからテキストをすべて読み取り、文字列にして返します。

`stream#readchar()`

ストリームから一文字分のデータを読み取り、文字列にして返します。ファイルの末尾に到達した場合は `nil` を返します。

`stream#seek(offset:number, origin?:symbol):reduce`

ストリームのシーク位置を先頭から `offset` の位置に動かします。`origin` に`cur を設定すると、現在の

シーク位置に offset だけ足した位置に移動します。このとき、offset にはマイナスの値を設定することができます。

stream#setencoding(encoding:string, dos_flag?:boolean)

ストリームのエンコーディングを指定します。引数encodingは、ストリームを文字列として読み書きするときの文字エンコーディング名です。dos_flag を true に設定すると、改行コードを以下のように変換します。

入力時: CR-LF を LF に変換

出力時: LF を CR-LF に変換

stream#tell()

ストリームのシーク位置を返します。

stream#write(buff:binary):reduce

ストリームにバイナリデータを書き込みます。

stream#copyfrom(src:stream:r, bytesunit:number => 65536):map:reduce {block?}

入力用ストリーム srcからストリームインスタンスにデータをコピーします。コピー処理はデフォルトで 65536 バイト単位で行われますが、引数 bytesunit でこれを変更することができます。

block を指定すると、コピー単位ごとにブロックパラメータを|buff:binary| という形式でブロックに渡して評価します。buff は入力用ストリームから入力したデータが入り、評価結果が binary 型の値であればそれを出力用ストリームに出力します。それ以外の型の場合はもとのデータを出力します。

stream#copyto(dst:stream:w, bytesunit:number => 65536):map:reduce {block?}

ストリームインスタンスから出力用ストリーム dst にデータをコピーします。コピー処理はデフォルトで 65536 バイト単位で行われますが、引数 bytesunit でこれを変更することができます。

block を指定すると、コピー単位ごとにブロックパラメータを|buff:binary| という形式でブロックに渡して評価します。buff は入力用ストリームから入力したデータが入り、評価結果が binary 型の値であればそれを出力用ストリームに出力します。それ以外の型の場合はもとのデータを出力します。

4.22.4. インスタンスプロパティ

すべての stream インスタンスは以下のプロパティを持ちます。

プロパティ	型	R/W	内容
stat	any	R	ストリームの属性を返すインスタンスです。ストリームの種類によって内容が異なります。
name	string	R	ストリームの名前を返します
identifier	string	R	ストリームが属する集合の中で、このストリームを一意に識別できる文字列を返します。例えば、ファイルシステムのストリームならば絶対パス名に、zip ファイル中ならばアーカイブに記録した名前になります。
readable	boolean	R	読みこみ可能なストリームならば true を返します
writable	boolean	R	書きこみ可能なストリームならば true を返します
codec	codec	R	ストリームに登録されているコーデックオブジェクトを返します。

4.23. string クラス

4.23.1. インスタンスの生成

- コード中に文字列リテラルを記述すると、string インスタンスの生成になります。

4.23.2. インスタンスメソッド

`string#align(len:number, padding:string => ' '):map:[center,left,right]`

文字列の長さを引数 len で指定した文字数でそろえます。もとの文字列が指定の長さに満たない場合は、引数 padding で指定した文字で埋めます。このとき、文字列の位置をアトリビュートで指示することができ、:center で中央、:left で左詰め、:right で右詰めになります。文字列の長さが len 以上である場合はもとの文字列を返します。

`string#capitalize()`

先頭の文字がアルファベットの小文字のとき、これらを大文字に変換した結果を返します。

`string#chop(suffix:*string):[icase,eol]`

何も引数やアトリビュートをつけずに実行すると、文字列中の最後の一文字を取り除いた結果を返します。アトリビュート:eol をつけると、最後が改行記号のときのみ取り除きます。コードが CR-LF という連なりになっている場合は、これら 2 文字をとりのぞきます。

引数に文字列を指定すると、これらの文字列が最後にあらわれたときのみとりのぞきます。この文字列は複数指定することができます。アトリビュート:icase が指定されると、大文字と小文字を区別しません。また、アトリビュート:eol とともに実行した場合は、まず改行コードがあればそれをとりのぞき、その後指定文字列の除去を行います。

`string#decodeuri()`

URI 書式で処理ができるようにした文字列から通常の文字列にして返します。

`string#each():map:[utf8,utf32] {block?}`

文字列中の文字をとりだし、文字コードを数値として返すイテレータを生成します。アトリビュート:utf8 をつけると、UTF-8 コード、:utf32 をつけると、UTF-32 コードの数値を返します。

`string#eachline(nlines?:number):[chop] {block?}`

文字列を一行ずつ切り出して返すイテレータを生成します。引数 nlines を指定すると、切り出す行数をその行数までに限定します。デフォルトでは、一行の文字列中に改行コードを含みますが、アトリビュート:chop をつけると改行コードをとりのぞきます。

`string#encode(codec:codec)`

文字列を指定のコーデックで変換した結果を binary 型として返します。

`string#encodeuri()`

URI 書式で処理ができるようにした文字列を返します。

`string#endswith(suffix:string, endpos?:number):map:[rest,icase]`

文字列が suffix で終了している場合は true、それ以外は false を返します。アトリビュート:icase をつけると、大文字と小文字を区別しません。

アトリビュート:rest をつけると、文字列が suffix で終了している場合、それよりも前の文字列を返します。それ以外は nil を返します。

string#escapehtml()

HTML 書式で処理ができるようにした文字列を返します。

string#template(stream?:stream:w):[lasteol,noindent]

文字列中に記述されたスクリプトを評価し、文字列に挿入します。スクリプトは文字列中で "${" と "}" に挟んで記述します。

string#find(sub:string, pos:number => 0):map:[rev,icase]

文字列 sub が見つかった文字位置を返します。見つからない場合は nil を返します。引数 pos を指定すると、その位置から文字列を探します。アトリビュート :rev をつけると、後尾から文字列を探します。アトリビュート:icase をつけると、大文字と小文字の区別をつけません。

string#format(values*):map

文字列に記述された printf のフォーマットに従って引数 values の値を埋め込んだ文字列を返します。

string#isempty()

文字列が空のとき true、それ以外は false を返します。

string#left(len?:number):map

左から指定文字数だけ取り出した文字列を返します。

string#len()

文字列中の *文字数* を返します。バイト数でないことに注意してください。

string#lower()

アルファベットを小文字に変換した結果を返します。

string#mid(pos:number => 0, len?:number):map

引数 pos の位置から長さ len 文字数だけ取り出した文字列を返します。

string#print(stream?:stream:w):void

文字列を引数 stream で指定したストリームに出力します。

string#println(stream?:stream:w):void

文字列を引数 stream で指定したストリームに出力し、最後に改行を出力します。

string#reader() {block?}

文字列内の文字コードを 1 バイトずつとりだすストリームを返します。文字コードは UTF-8 です。

string#replace(sub:string, replace:string, count?:number):map:[icase]

引数 sub で指定した文字列を replace に置き換えます。引数 count を指定すると、置き換える回数を限定します。アトリビュート:icase をつけると大文字と小文字を区別しません。

```
string#right(len?:number):map
```

右から指定文字数だけ取り出した文字列を返します。

```
string#split(sep?:string, count?:number):[icase] {block?}
```

引数 sep を境界にして文字列を切り分けた結果を返すイテレータを生成します。引数 count を指定すると、切り分ける数を限定します。アトリビュート:icase をつけると大文字と小文字を区別しません。

```
string#startswith(prefix:string, pos:number => 0):map:[rest,icase]
```

文字列が prefix で始まっている場合は true、それ以外は false を返します。アトリビュート:icase をつけると、大文字と小文字を区別しません。

アトリビュート:rest をつけると、文字列が suffix で始まっている場合、それよりも後の文字列を返します。それ以外は nil を返します。

```
string#strip():[both,left,right]
```

文字列の左右にある空白や改行要素をとりのぞいた結果を返します。

アトリビュート:left をつけると、左側のみの空白・改行要素をとりのぞきます。アトリビュート:right をつけると、右側のみとりのぞきます。アトリビュート:both は両側の空白・改行要素をとりのぞき、これがデフォルトの動作になります。

```
string#unescapehtml()
```

HTML 書式で処理ができるようにした文字列から通常の文字列にして返します。

```
string#upper()
```

アルファベットを大文字に変換した結果を返します。

```
string#zentohan()
```

文字列中の「全角文字」を、対応する ASCII 文字に変換した結果を返します。

4.24. operator クラス

4.24.1. インスタンスの生成

```
operator(op:symbol):map {block?}
```

指定のシンボルに対応した operator インスタンスを返します。

4.24.2. インスタンスメソッド

```
operator#assign(type_l:expr, type_r?:expr):map:void {block}
```

指定のオペレータ演算子に手続きを割り当てます。

5. sys モジュール

5.1. 概要

Gura 本体の動作モードを変えたり、実行状態を得たりするモジュールです。組み込みモジュールなので、インポートをしないで使用することができます。

5.2. モジュール関数

`sys.echo(flag:boolean)`

対話モードのとき、結果をエコーバックするか否かを設定します。`flag` に `true` を指定するとエコーバックが有効になります。

`sys.exit(status?:number)`

プログラムを終了します。`status` で終了コードを指定します。省略すると 0 になります。

5.3. モジュール変数

`sys` モジュール内には以下の変数があらかじめ設定されています。

変数	型	内容
`ps1`	`string`	対話モードで、インデントがかかっていないときのプロンプト
`ps2`	`string`	対話モードで、インデントがかかっている間のプロンプト
`stdin`	`stream`	標準入力に使うストリーム
`stdout`	`stream`	標準出力に使うストリーム
`stderr`	`stream`	標準エラー出力に使うストリーム
`path`	`list`	モジュールのサーチパスを記述したリスト
`version`	`string`	バージョン番号
`build`	`symbol`	Gura をビルドした環境のシンボル値 `` `gcc `` GNU C compiler `` `msc `` Microsoft Visual C++
`platform`	`symbol`	動作しているプラットフォームのシンボル値 `` `linux `` Linux `` `mswin `` Microsoft Windows
`executable`	`string`	実行ファイルのフルパス名
`datadir`	`string`	データディレクトリのフルパス名
`libdir`	`string`	ライブラリディレクトリのフルパス名
`argv`	`list`	引数リスト。`argv[0]` にはスクリプトの名前がフルパスで格納される

6. fs モジュール

6.1. 概要

ファイルシステムの操作をするモジュールです。組み込みモジュールなので、インポートをしないで使用することができます。

6.2. ストリームのオープン

open 関数でファイルシステム上のファイルをオープンできるようになります。
ストリームを受け取る引数に、ファイルシステム上のファイルパス名を指定できるようになります。

6.3. パスのサーチ

path.dir, path.walk, path.glob 関数で、ファイルシステム上のディレクトリパスをサーチできるようになります。

6.4. モジュール関数

```
fs.chdir(pathname:string) {block?}
```

カレントディレクトリを設定します。ブロックを指定した場合、カレントディレクトリを設定した後にブロックの内容を評価し、元のディレクトリに戻ります。

```
fs.getcwd()
```

カレントディレクトリを取得します。

```
fs.mkdir(pathname:string):map:void:[tree]
```

ディレクトリを作成します。
デフォルトでは、引数 pathname のパス名が複数の階層にまたがっていて、途中のディレクトリが存在しないとエラーになります。
アトリビュート :tree をつけると、途中のディレクトリが存在しないときそれらのディレクトリも作成します。

```
fs.rmdir(pathname:string):map:void:[tree]
```

ディレクトリを削除します。
デフォルトでは、削除対象のディレクトリ内にファイルや子ディレクトリが存在しているとエラーになります。
アトリビュート :tree をつけると、削除対象のディレクトリに含まれるファイルや子ディレクトリもすべて削除します。

```
fs.remove(pathname:string):map:void
```

ファイルを削除します。

```
fs.rename(src:string, dst:string):map:void
```

ファイルまたはディレクトリの名前を変更します。

```
fs.chmod(mode:number, pathname:string):void
```

ファイルの属性を数値で設定します。属性は数値のビット位置に対応しており、ビットを 1 に設定するとその属性が有効になります。ビット位置と属性の関係を以下に示します。

b8 b7 b6　　所有者のリード、ライト、実行属性

b5 b4 b3　　グループのリード、ライト、実行属性

b2 b1 b0　　その他のユーザのリード、ライト、実行属性

`fs.chmod(mode:string, pathname:string):void`

ファイルの属性を文字列で設定します。受け付ける文字列のフォーマットを正規表現であらわすと以下のようになります。

```
[ugoa]*([-+=][rwx]+)+
```

最初に、属性を設定する対象を指定し、続けて設定方法と属性を指定します。

設定する対象	
u	所有者
g	グループ
o	その他のユーザ
a	全てのユーザ

設定方法	
-	属性をとりのぞく
+	属性を加える
=	属性を設定する

属性	
r	リード属性
w	ライト属性
x	実行属性

6.5. fs.stat クラス

6.5.1. インスタンスプロパティ

関数 open が返すストリームが fs モジュールのものであるとき、このストリームインスタンスは stat という名前のプロパティを持っており、これは fs.stat 型のインスタンスです。このインスタンスは以下のプロパティを持ちます。

プロパティ	データ型	R/W	内容
pathname	string	R	dirname と filename をあわせたフルパス名
dirname	string	R	ディレクトリ名
filename	string	R	ファイル名
size	number	R	ファイルサイズのバイト数
uid	number	R	ユーザ ID
gid	number	R	グループ ID
atime	datetime	R	アクセス時刻
mtime	datetime	R	修正時刻
ctime	datetime	R	作成時刻
isdir	boolean	R	ディレクトリのとき true
ischr	boolean	R	キャラクタデバイスのとき true
isblk	boolean	R	キャラクタデバイスのとき true
isreg	boolean	R	通常ファイルのとき true
isfifo	boolean	R	FIFO のとき true

islnk	boolean	R	リンクファイルのとき true
issock	boolean	R	ソケットのとき true

7. os モジュール

7.1. 概要

OSの操作をまとめたモジュールです。組み込みモジュールなので、インポートをしないで使用することができます。

7.2. モジュール変数

os モジュール内には以下の変数があらかじめ設定されています。

変数	型	内容
stdin	stream	標準入力。デフォルトで sys.stdin が割り当てられています。
stdout	stream	標準出力。デフォルトで sys.stdout が割り当てられています。
stderr	stream	標準エラー出力。デフォルトで sys.stderr が割り当てられています。

7.3. モジュール関数

os.exec(pathname:string, args*:string):map:[fork]

外部実行可能ファイルを実行します。引数pathnameに実行可能ファイルのファイル名、argsに引数を指定します。

デフォルトでは、この関数は実行可能ファイルが終了するのを待ち、実行結果のエラーレベル（C 言語のプログラムならば main 関数の戻り値または exit 関数の引数値）を戻り値として返します。このとき、標準出力および標準エラー出力の内容をそれぞれ os.stdout と os.stderr に指定したストリームに対して出力します。

アトリビュート:forkをつけると、実行可能ファイルを起動した後、すぐに関数から処理が戻ります。この場合、戻り値は常に 0 です。

os.fromnative(buff:binary):map

OS 依存の文字列をスクリプトで処理できる文字列に変換します。

os.getenv(name:string):map

引数 name に対応する環境変数の値を文字列で返します。環境変数が設定されていない場合は空の文字列を返します。

os.putenv(name:string, value:string):void

引数 name に対応する環境変数の値を value に設定します。

os.redirect(stdin:stream:nil:r, stdout:stream:nil:w,
stderr?:stream:w) {block?}

標準入力 os.stdin、標準出力 os.stdout および標準エラー出力 os.stderr を指定の stream インスタンスに設定します。引数 stderr は省略可能で、省略すると stdout に指定したのと同じ stream インスタンスに設定します。

stdin に nil を設定すると、標準入力に何も接続しません。stdout や stderr に nil を設定すると、

これらの出力を抑止することができます。

ブロックを指定して実行すると、ストリームを設定してからブロックを評価し、評価後に設定をもとにもどします。

`os.redirect` の戻り値はブロックが指定されている場合はその評価値、指定されていなければ常に `nil` です。

`os.tonative(str:string):map`

スクリプトで処理できる文字列から OS 依存の文字列に変換します。

8. path モジュール

8.1. 概要

パス操作をまとめたモジュールです。組み込みモジュールなので、インポートをしないで使用することができます。

8.2. モジュール関数

path.absname(name:string):map:[http]

絶対パス名を返します。パス名は整形された形式で生成されます。整形の方法については path.regulate を参照ください。

path.bottom(pathname:string):map

パス名をパスセパレータで区切った時の最後の要素名を返します。

path.cutbottom(pathname:string):map

パス名をパスセパレータで区切った時の最後の要素名を取り除いた結果を返します。

path.dir(pathname?:string, pattern*:string):map:flat:[stat,icase,file,dir] {block?}

ディレクトリを表すパス名を指定し、含まれるファイルまたはディレクトリをサーチします。

引数 pathname はパス名です。引数 pattern には、ファイルまたはディレクトリのベース名に対するパターンを 0 個以上指定します。この引数を省略すると、すべてのファイルまたはディレクトリをサーチします。

アトリビュート:stat をつけるとパス名ではなく詳細情報を含んだ stat 型オブジェクトを返します。:icase は、パターンマッチングの際に大文字と小文字の区別をなくすアトリビュートです。:file や:dir をつけると、サーチ対象をそれぞれファイルまたはディレクトリに限定できます。

ブロック式をつけると、各サーチ結果ごとにブロックが繰り返し評価されます。このとき、ブロックには |pathname:string, idx:number| という形式で引数が渡されます。pathname はサーチ結果のパス名、idx はループのインデクス番号です。

path.dirname(pathname:string):map

パス名からディレクトリ名要素を抽出します。

path.exists(pathname:string):map

指定したパスが存在するとき true を返します。それ以外のときは false を返します。

path.filename(pathname:string):map

パス名からファイル名要素を抽出します。

path.glob(pattern:string):map:flat:[stat,icase,file,dir] {block?}

パターンに適合するファイルやディレクトリをサーチします。

引数 pattern にパターンを指定します。このパターンはディレクトリ名を含むことができ、パス名の途中のディレクトリ名にもワイルドカードを使えます。

アトリビュート:stat をつけるとパス名ではなく詳細情報を含んだ stat 型オブジェクトを返しま

す。:icase は、パターンマッチングの際に大文字と小文字の区別をなくすアトリビュートです。:file や:dir をつけると、サーチ対象をそれぞれファイルまたはディレクトリに限定できます。

ブロック式をつけると、各サーチ結果ごとにブロックが繰り返し評価されます。このとき、ブロックには |pathname:string, idx:number| という形式で引数が渡されます。pathname はサーチ結果のパス名、idx はループのインデクス番号です。

path.join(paths+:string):map:[uri]

パス名をつなぎあわせた結果を返します。

つなぎあわせるときのパスセパレータは、現在動作している OS が Windows 系の場合はバックスラッシュ "¥"、それ以外の場合はスラッシュ "/" を使用します。ただし、アトリビュート:uri を指定するとパスセパレータとして常にスラッシュ "/" を使用します。

path.match(pattern:string, name:string):map:[icase]

文字列 name がファイル名マッチングパターン pattern に合致しているとき true を返します。それ以外は false を返します。デフォルトでは比較文字列の大文字と小文字を区別しますが、アトリビュート :icase をつけると区別しません。

マッチングパターンには以下のワイルドカードを使用することができます。

ワイルドカード	説明
*	任意の長さの文字列
?	任意の一文字
[…]	ブラケット内で指定した文字のいずれか
[!…]	ブラケット内で指定した文字以外のいずれか

path.regulate(pathname:string):map:[uri]

パス名を以下の条件に従って整形します。

- パスセパレータを統一します。現在動作している OS が Windows 系の場合はバッククォーテーション "¥"、それ以外はスラッシュ "/" を使用します。ただし、アトリビュート:uri を指定すると常にパスセパレータとしてスラッシュ "/" を使用します。
- 相対パス指定 "." をとりのぞきます。
- 相対パス指定 ".." をとりのぞき、ひとつ上のパス要素を削除します。

path.split(pathname:string):map:[bottom]

パス名をディレクトリ名とファイル名に分離し、リストにして返します。これは、path.dirname と path.filename の結果をあわせたものと同じです。

アトリビュート:bottom をつけると、パスセパレータで区切った時の前の要素と最後の要素をリストにして返します。これは、path.cutbottom と path.bottom の結果をあわせたものと同じです。

path.splitext(pathname:string):map

パス名のサフィックスを分離し、分離した前の部分とサフィックスをリストにして返します。

path.stat(pathname:string):map

指定したパスの属性を収めた stat インスタンスを生成して返します。stat インスタンスの内容はパス名を解釈したモジュールによって異なります。

```
path.walk(pathname?:string, maxdepth?:number, pattern*:string)
                                    :map:flat:[stat,icase,file,dir] {block?}
```

パス名で指定したディレクトリを基点として含まれるファイルまたはディレクトリを再帰的にサーチします。

引数 pathname はパス名です。maxdepth には、サーチするディレクトリの深さを指定します。0 を指定すると基点のディレクトリのみのサーチとなり、これは path.dir の動作と同じになります。省略すると、深さの制限がなくなります。

引数 pattern には、ファイルまたはディレクトリのベース名に対するパターンを 0 個以上指定します。この引数を省略すると、すべてのファイルまたはディレクトリをサーチします。

アトリビュート:stat をつけるとパス名ではなく詳細情報を含んだ stat 型オブジェクトを返します。:icase は、パターンマッチングの際に大文字と小文字の区別をなくすアトリビュートです。:file や:dir をつけると、サーチ対象をそれぞれファイルまたはディレクトリに限定できます。

ブロック式をつけると、各サーチ結果ごとにブロックが繰り返し評価されます。このとき、ブロックには |pathname:string, idx:number| という形式で引数が渡されます。pathname はサーチ結果のパス名、idx はループのインデクス番号です。

9. math モジュール

9.1. 概要

数学演算処理をまとめたモジュールです。組み込みモジュールなので、インポートをしないで使用することができます。

9.2. モジュール関数

`math.abs(num):map`

絶対値を計算します。

`math.acos(num):map:[deg]`

アークコサインを計算し、角度をラジアン値で返します。アトリビュート:deg をつけると、degree 値で結果を返します。

`math.arg(num):map:[deg]`

num が複素数のとき、極座標における偏角をラジアン値で返します。アトリビュート:deg をつけると、degree 値で結果を返します。

`math.asin(num):map:[deg]`

アークサインを計算し、角度をラジアン値で返します。アトリビュート:deg をつけると、degree 値で結果を返します。

`math.atan(num):map:[deg]`

アークタンジェントを計算し、角度をラジアン値で返します。アトリビュート:deg をつけると、degree 値で結果を返します。

`math.atan2(num1, num2):map:[deg]`

num1 / num2 の値に対するアークタンジェントを計算し、角度をラジアン値で返します。num2 が 0 のときの値は、num1 が正のとき π/2、負のとき −π/2 になります。アトリビュート:deg をつけると、degree 値で結果を返します。

`math.bezier(nums[]+:number)`

`math.ceil(num):map`

小数点以下一位を切り上げた数値を返します。

`math.conj(num):map`

共役複素数を返します。

`math.cos(num):map:[deg]`

コサインを計算します。指定する角度の単位はラジアンです。アトリビュート:deg をつけると、角度を degree 値で指定できます。

math.cosh(num):map

ハイパボリックコサインを計算します。指定する角度の単位はラジアンです。

math.covariance(a:iterator, b:iterator)

二つのイテレータ要素間の共分散値を計算します。

math.cross_product(a[], b[])

math.diff(expr:expr, var:symbol):map

expr が数学の式からなるとき、var を変数とした微分演算処理を行い、結果を expr インスタンスで返します。

math.dot_product(a[], b[])

math.exp(num):map

底が e のべき乗値を計算します。

math.fft(seq[])

t.b.d

math.floor(num):map

小数点以下一位を切り捨てた数値を返します。

math.imag(num):map

num が複素数のとき、虚数成分を返します。それ以外の場合 0 を返します。

math.integral()

t.b.d

math.least_square(x:iterator, y:iterator, dim:number => 1, var:symbol => `x)

与えられた x, y 列に対し、最小二乗法による近似式を計算し、その演算式を持った function インスタンスを生成します。インスタンスの名前を f としたとき、呼び出し形式は以下のようになります。

```
f(x:number):map
```

引数 dim で近似式の次数を指定します。デフォルトでは一次式による近似を行います。

引数 var は、生成する function インスタンスの引数のシンボルを指定します。デフォルトは`x です。

math.log(num):map

底が e の log 値を計算します。

math.log10(num):map

底が 10 の log 値を計算します。

math.norm(num):map

ノルムを計算します。

`math.optimize(expr:expr):map`

`expr` が数学の式からなるとき、フォーマットを最適化した結果を `expr` インスタンスで返します。

`math.real(num):map`

`num` が複素数のとき、実数成分を返します。それ以外の場合 `num` そのものを返します。

`math.sin(num):map:[deg]`

サインを計算します。指定する角度の単位はラジアンです。アトリビュート`:deg` をつけると、角度を degree 値で指定できます。

`math.sinh(num):map`

ハイパボリックサインを計算します。指定する角度の単位はラジアンです。

`math.sqrt(num):map`

平方根を計算します。

`math.tan(num):map:[deg]`

タンジェントを計算します。指定する角度の単位はラジアンです。アトリビュート`:deg` をつけると、角度を degree 値で指定できます。

`math.tanh(num):map`

ハイパボリックタンジェントを計算します。指定する角度の単位はラジアンです。

10. time モジュール

10.1. 概要

時刻操作をまとめたモジュールです。組み込みモジュールなので、インポートをしないで使用することができます。

10.2. モジュール関数

`time.clock()`

1/100 秒ごとに数値が増えるシステムクロック値を返します。

`time.isleap(year:number):map`

指定した年がうるう年のとき `true` を返します。それ以外は `false` を返します。

`time.monthdays(year:number, month:number):map`

西暦と月を受け取り、その月の最終日を返します。

`time.parse(str:string):map`

`time.sleep(secs:number)`

指定した秒数だけスリープします。

`time.weekday(year:number, month:number, day:number):map`

指定した日の曜日をインデクス値で返します。日曜日が 0 で土曜日が 6 になります。

10.3. モジュール変数

変数	型	内容
Sunday	number	日曜日を表すインデクス番号 0 が代入されています
Monday	number	月曜日を表すインデクス番号 1 が代入されています
Tuesday	number	火曜日を表すインデクス番号 2 が代入されています
Wednesday	number	水曜日を表すインデクス番号 3 が代入されています
Thursday	number	木曜日を表すインデクス番号 4 が代入されています
Friday	number	金曜日を表すインデクス番号 5 が代入されています
Saturday	number	土曜日を表すインデクス番号 6 が代入されています

10.4. datetime クラス

10.4.1. 概要

時刻を表すクラスです。

10.4.2. インスタンスの生成

```
time.datetime(year:number => 0, month:number => 1, day:number => 1,
              hour:number => 0, min:number => 0, sec:number => 0,
              usec:number => 0, minsoff?:number):map
```

年月日および時刻を指定した datetime インスタンスを生成します。

```
time.now():[utc]
```

現在の年月日および時刻が入った datetime インスタンスを生成します。

```
time.time(hour:number => 0, minute:number => 0,
      sec:number => 0, usec:number => 0):map
```

時刻を設定した datetime インスタンスを生成します。年月日は 0 年 1 月 1 日に設定されます。

```
time.today():[utc]
```

今日の日付が入った datetime インスタンスを生成します。時刻は 00:00:00 が設定されます。

10.4.3. インスタンスメソッド

```
datetime#format(format => `w3c)
```

指定のフォーマットで日時データを文字列に変換します。引数 format にはシンボルまたは文字列を指定します。引数 format にシンボルを指定した場合、以下のように変換します。

シンボル	説明
`w3c	W3C の仕様で使われる日時フォーマットに変換します。 例: 2010-11-06T08:49:37Z
`http	RFC で定義される HTTP の仕様で使われる日時フォーマットに変換します。 例: Sat, 06 Nov 2010 08:49:37 GMT
`asctime	C 言語の asctime 関数のフォーマットで変換します。 例: Sat Nov 6 08:49:37 +0000 2010

引数 format に文字列を指定した場合、以下の指定子で日時データの要素を文字列変換します。指定子以外の文字はそのまま文字列に挿入されます。

指定子	説明
%d	日
%H	時間（24 時間制）
%I	時間（12 時間制）
%m	月
%M	分
%S	秒
%w	日曜日を 0 とした曜日のインデクス番号
%y	年の下 2 桁

%Y	年

datetime#settzoff(mins:number):reduce

UTC からの時差を分単位で指定します。

datetime#clrtzoff():reduce

UTC からの時差情報を取り除きます。

datetime#utc()

日時を UTC に変換した結果を返します。

10.4.4. インスタンスプロパティ

プロパティ	型	R/W	内容
year	number	R/W	西暦
month	number	R/W	1 から 12 までの数値で 1 月から 12 月を表します。
day	number	R/W	1 から 31 までの数値で 1 日から 31 日を表します。
hour	number	R/W	0 から 23 までの数値で 0 時から 23 時を表します。
min	number	R/W	0 から 59 までの数値で 0 分から 59 分を表します。
sec	number	R/W	0 から 59 までの数値で 0 秒から 59 秒を表します。
usec	number	R/W	0 から 999 までの数値で 0 ミリ秒から 59 ミリ秒を表します。
wday	number	R	0 から 6 までの数値で日曜日から土曜日を表します。
week	symbol	R	週の名前を以下のシンボルで表します。 `sunday, `monday, `tuesday, `wednesday, `thursday, `friday, `saturday
yday	number	R	1 から 366 までの数値で年の初めからの日数を表します。
unixtime	number	R	UTC の 1970 年 1 月 1 日 00:00:00 からの経過時間を秒で表わします

10.5. timedelta クラス

10.5.1. 概要

時刻間の差を表すクラスです。以下の状況で使用します。

- datetime インスタンス間の差を計算したときの結果
- datetime インスタンスの時刻を増減させるときの差分

10.5.2. インスタンスの生成

time.delta(days:number => 0, secs:number => 0, usecs:number => 0)

指定した値を持つ timedelta インスタンスを生成します。

10.5.3. インスタンスプロパティ

プロパティ	型	R/W	内容
days	number	R/W	0 からの数値で日数を表します。
secs	number	R/W	0 から 86399 (60×60×24 - 1) までの数値で秒を表します。
usecs	number	R/W	0 から 999999 までの数値でマイクロ秒を表します。

11. conio モジュール

11.1. 概要

コンソール操作をまとめたモジュールです。組み込みモジュールなので、インポートをしないで使用することができます。

11.2. モジュール関数

conio.clear(region?:symbol):void

コンソール画面の内容を消去します。

conio.getwinsize()

コンソールサイズを [width, height] というリスト形式で返します。

conio.moveto(x:number, y:number):map:void {block?}

カーソルを指定の位置に移動します。ブロックを指定すると、カーソル移動後にそのブロックの内容を評価し、その後に元のカーソル位置を復元します。

conio.setcolor(fg:symbol:nil, bg?:symbol):map:void {block?}

テキストの前景色と背景色をシンボルで指定します。ブロックを指定すると、色を変えた後にそのブロックの内容を評価し、その後に元のテキスト色を復元します。指定できるシンボルは以下のとおりです。

`black	`gray
`blue	`bright_blue
`green	`bright_green
`aqua / `cyan	`bright_aqua / `bright_cyan
`red	`bright_red
`purple / `magenta	`bright_purple / `bright_magenta
`yellow	`bright_yellow
`white	`bright_white

conio.waitkey():[raise]

コンソールでキーが押されるのを待ち、入力された文字コードを返します。アトリビュート raise を指定すると、Ctrl+C が入力されたときに Terminate シグナルを発行し、インタープリターの動作を中断します。

特殊キーの文字コードとして、以下の値が定義されています。

K_BACKSPACE, K_DELETE, K_DOWN, K_END, K_ESCAPE

K_HOME, K_INSERT, K_LEFT, K_PAGEDOWN, K_PAGEUP

K_RETURN, K_RIGHT, K_SPACE, K_TAB, K_UP

12. hash モジュール

12.1. 概要

ハッシュ値を計算するモジュールです。使用するには import 関数を使って hash モジュールをインポートします。

ハッシュ値を計算するには、hash ストリームオブジェクトを生成した後 hash#write() または hash#update() でデータ列を入力し、最後に hash#digest プロパティまたは hash#hexdigest プロパティを参照して結果を得ます。

12.2. モジュール関数

hash.crc32(stream?:stream:r) {block?}

CRC32 値を算出するストリームを作成して返します。引数 stream を指定すると、そのデータ内容を生成したストリームに書き込みます。

hash.md5(stream?:stream:r) {block?}

MD5 値を算出するストリームを作成して返します。引数 stream を指定すると、そのデータ内容を生成したストリームに書き込みます。

hash.sha1(stream?:stream:r) {block?}

SHA1 値を算出するストリームを作成して返します。引数 stream を指定すると、そのデータ内容を生成したストリームに書き込みます。

13. http モジュール

13.1. 概要

HTTP プロトコルのサーバとクライアント処理を提供するモジュールです。実装は RFC2616 で定められる仕様に基づきます。使用するには import 関数を使って http モジュールをインポートします。
以下の URL で公開されている zlib ライブラリを内部で使用しています。

http://zlib.net/

13.2. パス名の拡張

パス名が "http:" で始まっていると、http モジュールによってパスやストリームを処理します。

この拡張により、以下の操作が可能になります。

- open 関数で HTTP プロトコルを通じたファイルをオープンできるようになります。
- ストリームを受け取る引数に、HTTP のファイルパス名を指定できるようになります。

13.3. モジュール変数

変数	型	内容
proxies	http.proxy[]	http.addproxy で追加した http.proxy インスタンスのリストです

13.4. モジュール関数

```
http.addproxy(addr:string, port:number,
                              userid?:string, password?:string) {criteria?}
```

引数 addr と port に HTTP プロキシサーバのアドレスおよびポート番号を指定して HTTP プロキシ http.proxy インスタンスを生成し、モジュール変数 net.proxies に追加します。
プロキシサーバの接続で認証が必要な場合、userid と password を指定します。
ブロック criteria を省略すると、HTTP にクライアントとしてアクセスしたとき、常にこのメソッドで指定したプロキシをデフォルトとして使用します。
criteria をつけると、ブロックの評価結果が true のときのみこのプロキシを使います。criteria にはブロックパラメータが |addr:string| という形式で渡されます。addr はアクセス先のアドレスです。criteria は addr の内容をもとに、このプロキシを通するべきか判断します。

```
http.parsequery(query:string)
```

クエリー文字列をパースし、得られたキーと値を格納した dict インスタンスを返します。

```
http.splituri(uri:string)
```

URI を以下のようなフィールドに分けたリストを返します。

```
[scheme, authority, path, query, fragment]
```

該当するフィールドが URI 中に無い場合は空の文字列がその位置に入ります。

```
http.uri(scheme:string, authority:string,
                              path:string, query?:string, fragment?:string)
```

フィールドをもとにして URI を構成します。必要のないフィールドには空の文字列を入れます。

13.5. http.server クラス

13.5.1. インスタンスの生成

```
http.server(addr?:string, port:number => 80) {block?}
```

指定したアドレスおよびポート番号で HTTP リクエストを待ちうける http.server インスタンスを生成します。

13.5.2. インスタンスプロパティ

プロパティ	データ型	R/W	説明
sessions	http.session[]	R	この server インスタンスが保持している session インスタンスのリスト

13.5.3. インスタンスメソッド

```
http.server#wait() {block?}
```

HTTP リクエストを待ち受けます。リクエストが来たらその内容を格納した http.request インスタンスを生成します。

ブロックを指定した場合、ブロックパラメータを |req:http.request| という形式で渡します。ブロックの評価が終わると、再び HTTP リクエストの待ち受けに戻ります。

13.5.4. サンプルプログラム

サーバプログラムの例を以下に示します。

```
import(http)
http.server(port => 80).wait {|req|
    html = '<html>hello world</html>'
    req.response('200', nil, html.encode('utf-8'),
            'Cache-Control'     => 'private'
            'Server'            => 'Test_HTTP_Server'
            'Connection'        => 'Keep-Alive'
            'Content-Type'      => 'text/html; charset:utf-8')
}
```

13.6. http.client クラス

13.6.1. インスタンスの生成

```
http.client(addr:string, port:number => 80,
                    addrProxy?:string, portProxy?:number,
                    useridProxy?:string, passwordProxy?:string) {block?}
```

指定したアドレスおよびポート番号に HTTP プロトコルで接続処理を行い、http.client インスタンス

を生成します。

引数 addrProxy および portProxy にプロキシサーバのアドレスおよびポート番号を指定すると、そのプロキシを通した HTTP アクセスを行います。認証が必要な場合は useridProxy と passwordProxy にユーザ名とパスワードを指定します。

プロキシの指定を省略すると、http.addproxy で登録したプロキシのうち条件に合致するものを探し、なければダイレクトで接続をします。

13.6.2. インスタンスメソッド

```
http.client#request(method:string, uri:string, body?:stream:r,
                        version:string => 'HTTP/1.1', header%) {block?}
```

接続したサーバに対してリクエストを発行し、受信したレスポンスをもとに http.response インスタンスを生成して返します。引数 method にリクエストのメソッド、uri にリクエスト URI を指定します。uri 文字列中にホスト名は含みません。

引数 body には、メッセージヘッダに続いて送信するメッセージボディのストリームを指定します。省略した場合、メッセージボディは送信しません。

名前付き引数として、'*field-name*'=>'*field-value*' の形式で引数リストに入れると、メッセージヘッダ中にそれらのフィールド定義を追加します。

block が定義されていると、生成された http.response インスタンスをブロックパラメータの引数にしてブロックを評価します。

以下にメソッドの使用例を示します。

```
http.client#cleanup()
```

レスポンスのメッセージボディをキャンセルするときに実行します。

13.6.3. リクエスト発行インスタンスメソッド

リクエストを発行するには http.client#request メソッドを使いますが、よく使われるリクエストについては、リクエストの名前を持ったインスタンスメソッドが用意されています。以下にメソッド名と http.client#request の method 引数に渡す文字列の一覧を示します。

メソッド名	method 引数に渡す文字列
http.client#options	'options'
http.client#get	'get'
http.client#head	'head'
http.client#post	'post'
http.client#put	'put'
http.client#delete	'delete'
http.client#trace	'trace'
http.client#connect	'connect'

13.6.4. サンプルプログラム

クライアントプログラムの例を以下に示します。

```
import(http)
http.client('hoge.com') {|c|
    resp = c.get('/',
        'Connection'    => 'keep-alive'
        'Keep-Alive'    => '300')
    resp.body.copyto(sys.stdout)
}
```

13.7. http.stat クラス

13.7.1. 概要

open 関数などで http モジュールを通したストリームを取得すると、ストリームインスタンス中に stat という名前の http.stat インスタンスが作成されます。

13.7.2. メッセージヘッダのフィールド定義

http.stat クラスのインスタンスが stat という名前の変数に割り当てられているとき、インデクスアクセス stat['*field-name*'] でメッセージヘッダのフィールドに定義されている値を得ることができます。
フィールドが存在しない場合、この値は nil になります。存在する場合、そのフィールド名に対して最後に定義された値を文字列で返します。

13.7.3. インスタンスプロパティ

フィールド定義の中で時刻に関するものについては適切なデータ型に変換したプロパティが用意されています。

プロパティ	データ型	R/W	対応するフィールド	RFC2616
date	datetime	R	Date	14.18
expires	datetime	R	Expires	14.21
last_modified	datetime	R	Last-Modified	14.29

13.7.4. インスタンスメソッド

http.stat#field(name:string):map:[raise]

フィールド定義の値を文字列のリストで返します。指定のフィールド定義が無い場合空のリストを返します。
アトリビュート:raise をつけると、指定のフィールド定義が無い場合エラーになります。

13.8. http.request クラス

13.8.1. 概要

http.server インスタンスで wait メソッドを実行したときの戻り値として生成されます。サーバプログラムは、http.request のプロパティの値やメッセージボディの内容を確認し、response または respchunk メソッド

で適切なレスポンスを返します。

13.8.2. メッセージヘッダのフィールド定義

http.request クラスのインスタンスが req という名前の変数に割り当てられているとき、インデクスアクセス req['*field-name*'] でメッセージヘッダのフィールドに定義されている値を得ることができます。
フィールドが存在しない場合、この値は nil になります。存在する場合、そのフィールド名に対して最後に定義された値を文字列で返します。

13.8.3. インスタンスプロパティ

http.request インスタンスが持っているプロパティは以下の通りです。

プロパティ	データ型	R/W	説明
method	string	R	リクエストメソッド
uri	string	R	リクエスト URI
scheme	string	R	リクエスト URI 中の scheme 要素
authority	string	R	リクエスト URI 中の authority 要素
path	string	R	リクエスト URI 中の path 要素
query	string	R	リクエスト URI 中の query 要素
fragment	string	R	リクエスト URI 中の fragment 要素
version	string	R	HTTP バージョン
body	stream	R	リクエストのメッセージボディを受信するストリーム
session	http.session	R	セッション情報

プロパティ scheme、authority、path、query および fragment は、プロパティ uri の文字列から抽出したものです。同じ結果はプロパティ uri の内容を http.spliturui 関数で分割して得ることができます。
session プロパティは、http.server インスタンスが保持している session インスタンスへの参照です。セッションが保持されているかぎり、このプロパティは常に同じインスタンスを指します。

フィールド定義の中で時刻に関するものについては適切なデータ型に変換したプロパティが用意されています。

プロパティ	データ型	R/W	対応するフィールド	RFC2616
date	datetime	R	Date	14.18
expires	datetime	R	Expires	14.21
last_modified	datetime	R	Last-Modified	14.29

13.8.4. インスタンスメソッド

http.request#field(name:string):map:[raise]

フィールド定義の値を文字列のリストで返します。指定のフィールド定義が無い場合空のリストを返します。
アトリビュート:raise をつけると、指定のフィールド定義が無い場合エラーになります。

```
http.request#response(code:string, reason?:string, body?:stream:r,
                              version:string => 'HTTP/1.1', header%):reduce
```

リクエストに対するレスポンスを送信します。このメソッドは、メッセージボディが必要ないか、レスポンスとして返すメッセージボディの長さがあらかじめ分かっているときに使用します。

引数 code に 3 桁の数字からなるステータスコード、reason にレスポンスの説明をするテキスト文字列を指定します。

引数bodyには、メッセージヘッダに続いて送信するメッセージボディのストリームを指定します。省略した場合、メッセージボディは送信しません。

名前付き引数として、'*field-name*'=>'*field-value*' の形式で引数リストに入れると、メッセージヘッダ中にそれらのフィールド定義を追加します。

```
http.request#respchunk(code:string, reason?:string,
                               version:string => 'HTTP/1.1', header%)
```

リクエストに対するレスポンスを送信し、出力用の stream インスタンスを生成します。このメソッドは、レスポンスとして返すメッセージボディの長さがあらかじめ分からない場合に使用します。

引数 code に 3 桁の数字からなるステータスコード、reason にレスポンスの説明をするテキスト文字列を指定します。

名前付き引数として、'*field-name*'=>'*field-value*' の形式で引数リストに入れると、メッセージヘッダ中にそれらのフィールド定義を追加します。

生成された stream インスタンスに対してメッセージボディのデータを書き込みます。stream#write メソッドを呼び出すごとに chunked-body を作成します。

```
http.request#ismethod(method:string)
```

リクエストのメソッド名を調べます。メソッドが引数 methodと等しければ true、それ以外は false を返します。

13.9. http.session クラス

13.9.1. 概要

クライアントとのセッション情報を保持するクラスです。メソッド http.server#wait で生成される http.request インスタンスのプロパティとして存在します。

http.session インスタンスは、セッションが持続している間は常に同じ実体を参照します。そのため、セッションで保持すべき変数やオブジェクトを http.session インスタンスのプロパティにして利用することができます。

13.9.2. インスタンスプロパティ

セッションに関する以下のインスタンスプロパティを取得できます。

プロパティ	データ型	R/W	説明
server	http. server	R	このセッションを保持している server インスタンス

remote_ip	string	R	リクエスト元の IP アドレス
remote_host	string	R	リクエスト元のホスト名
remote_logname	string	R	t.b.d
local_ip	string	R	t.b.d
local_host	string	R	t.b.d
date	datetime	R	セッションを開始した日時

13.10. http.response クラス

13.10.1. 概要

http.client インスタンスで request メソッドを実行したときの戻り値として生成されます。クライアントプログラムは、http.response の情報を見てレスポンスの状態を確認し、メッセージボディを受信します。

13.10.2. メッセージヘッダのフィールド定義

http.response クラスのインスタンスが resp という名前の変数に割り当てられているとき、インデクスアクセス resp['*field-name*'] でメッセージヘッダのフィールドに定義されている値を得ることができます。
フィールドが存在しない場合、この値は nil になります。存在する場合、そのフィールド名に対して最後に定義された値を文字列で返します。

13.10.3. インスタンスプロパティ

レスポンスのステータスを以下のプロパティで取得できます。

プロパティ	データ型	R/W	説明
version	string	R	HTTP バージョン
code	string	R	3 桁の数字からなるステータスコード
reason	string	R	レスポンスの説明を表すテキスト文字列
field_names	list	R	格納されているフィールド定義名のリスト
body	stream	R	レスポンスのメッセージボディを受信するストリーム

フィールド定義の中で時刻に関するものについては適切なデータ型に変換したプロパティが用意されています。

プロパティ	データ型	R/W	対応するフィールド	RFC2616
date	datetime	R	Date	14.18
expires	datetime	R	Expires	14.21
last_modified	datetime	R	Last-Modified	14.29

13.10.4. インスタンスメソッド

http.response#field(name:string):map:[raise]

フィールド定義の値を文字列のリストで返します。指定のフィールド定義が無い場合空のリストを返します。

アトリビュート:raise をつけると、指定のフィールド定義が無い場合エラーになります。

14. bmp モジュール

14.1. 概要

イメージデータを Microsoft BMP イメージフォーマットで読み書きするモジュールです。使用するには import 関数を使って bmp モジュールをインポートします。

14.2. ストリーム処理

image 関数で指定したストリームが以下のいずれかの条件に合致すると、それを BMP イメージデータと認識して読み込み、image インスタンスを生成します。

- ストリームの識別子にサフィックス .bmp がついている（大小文字の区別はなし）
- ストリームの先頭が "BM" で始まっている

image#write メソッドで指定したストリームが以下の条件に合致すると、BMP イメージデータを出力します。

- ストリームの識別子にサフィックス .bmp がついている（大小文字の区別はなし）

14.3. image クラスの拡張

14.3.1. インスタンスメソッド

image#bmpread(stream:stream:r):reduce

指定のストリームから BMP フォーマットのデータを読み込んで image インスタンスにデータを展開します。

image#bmpwrite(stream:stream:w):reduce

image インスタンスのデータを BMP フォーマットにして指定のストリームに書き込みます。

15. gif モジュール

15.1. 概要

イメージデータを GIF (Graphics Interchange Format) イメージフォーマットで読み書きするモジュールです。使用するには import 関数を使って gif モジュールをインポートします。
GIF89a の規格をサポートしており、アニメーション GIF を扱うことができます。実装は、以下の URL の記述に基づきます。

http://www.w3.org/Graphics/GIF/spec-gif89a.txt

15.2. ストリームの読み書き

image 関数で指定したストリームが以下のいずれかの条件に合致すると、それを GIF イメージデータと認識して読み込み、image インスタンスを生成します。

- ストリームの識別子にサフィックス .gif がついている（大小文字の区別はなし）
- ストリームの先頭が "GIF87a" もしくは "GIF89a" で始まっている

image#write メソッドで指定したストリームが以下の条件に合致すると、GIF イメージデータを出力します。

- ストリームの識別子にサフィックス .gif がついている（大小文字の区別はなし）

15.3. gif.content クラス

15.3.1. 概要

GIF ファイルは複数のイメージデータを格納できるフォーマットなので、単一の image インスタンスではファイル全体のデータ構造を処理することができません。gif.content クラスを使うと、複数のイメージデータを格納・参照したり、GIF フォーマットの詳細なパラメータの取得や設定ができるようになります。

15.3.2. GIF Data Stream の構造

gif.content クラスは以下の図に示す GIF Data Stream の構造を表します。オプショナルなブロックには色がつけられています。

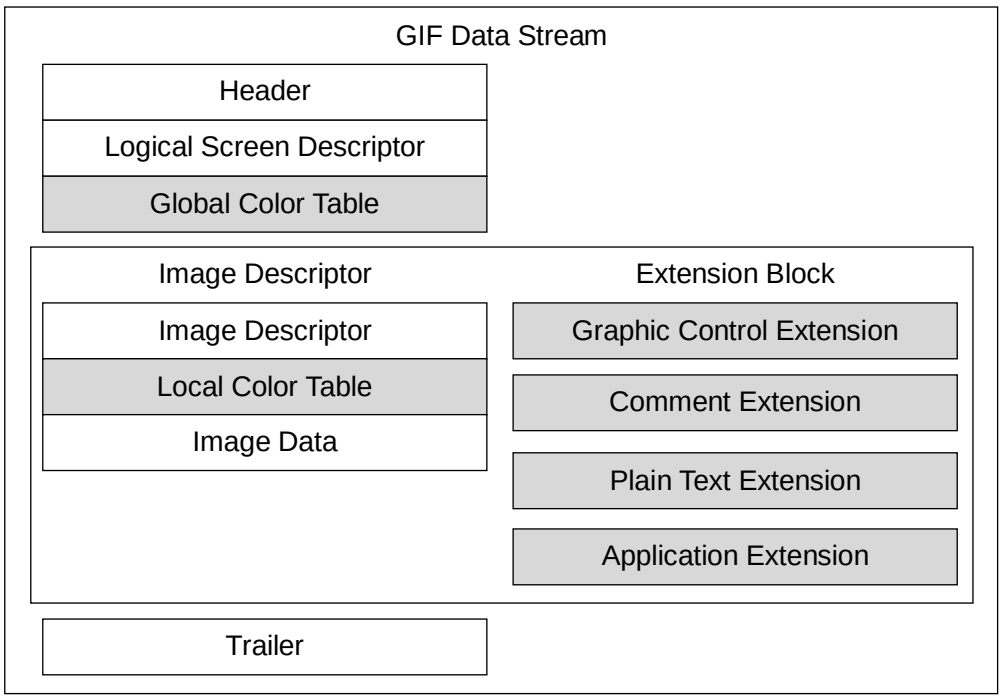


Image Descriptor と Extension Block は任意の数だけ GIF Data Stream に表れます。ただし、Graphic Control Extension は各 Image Descriptor の前に一つだけ置くことができます。

Image Data の内容は、image インスタンスのリストの形式で、gif.content 中に images という名前のプロパティとして格納されます。各 image インスタンスは Image Descriptor、Local Color Table および Graphic Control Extension の内容をプロパティとして持ちます。

Header、Logical Screen Descriptor、Comment Extension、Plain Text Extension および Application Extension の内容はそれぞれ gif.content 中に Header、LogicalScreenDescriptor、CommentExtension、PlainTextExtension および ApplicationExtension という名前のプロパティとして格納されます。

15.3.3. 制限事項

GIF の仕様では、Comment Extension、Plain Text Extension および Application Extension を複数格納することができますが、gif.content インスタンスで扱えるのはそれぞれ一個ずつです。複数存在する場合、最後に表れたデータを gif.content インスタンスに格納します。

15.3.4. インスタンスの生成

```
gif.content(stream?:stream:r, format:symbol => `rgba) {block?}
```

gif.content インスタンスを生成します。引数 stream を指定すると、そのストリームから GIF ファイル形式のデータを読み込みます。引数 format は、内部に保持する image インスタンスのフォーマットを指定します。

15.3.5. インスタンスメソッド

```
gif.content#addimage(image:image, delayTime:number => 0, leftPos:number => 0,
                topPos:number => 0, disposalMethod:symbol => `none):map:reduce
```

gif.content インスタンスにイメージデータを追加します。

gif.content#write(stream:stream:w):reduce

gif.content インスタンスの内容を GIF ファイル形式でストリームに書き込みます。

15.3.6. インスタンスプロパティ

gif.content インスタンスは、images というプロパティを持ち、これは image インスタンスのリストになっています。

また、gif.content インスタンスは GIF フォーマットの内部データを表す以下のプロパティを持っています。

プロパティ	データ型	R/W	定義
Header	gif.Header	R	required
LogicalScreenDescriptor	gif.LogicalScreenDescriptor	R	required
CommentExtension	gif.CommentExtension	R	optional
PlainTextExtension	gif.PlainTextExtension	R	optional
ApplicationExtension	gif.ApplicationExtension	R	optional

CommentExtension、PlainTextExtension および ApplicationExtension はオプショナルな情報で、GIF フォーマット中に存在しない場合、これらのプロパティは nil になります。

15.3.7. インスタンスプロパティの詳細

プロパティ gif.content#Header は gif.Header クラスのインスタンスで、GIF フォーマット中の Header の内容を表します。以下のプロパティを持っています。

プロパティ	データ型	R/W	内容
Signature	binary	R	GIF データの先頭を表す "GIF" というデータが入ります。
Version	binary	R	GIF のバージョンが入ります。"87a" か"89a" になります。

プロパティ gif.content#LogicalScreenDescriptor は gif.LogicalScreenDescriptor クラスのインスタンスで、GIF フォーマット中の Logical Screen Descriptor の内容を表します。以下のプロパティを持っています。

プロパティ	データ型	R/W	内容
LogicalScreenWidth	number	R	論理スクリーンにおけるイメージの幅をピクセル単位で表わします。
LogicalScreenHeight	number	R	論理スクリーンにおけるイメージの高さをピクセル単位で表わします。
GlobalColorTableFlag	boolean	R	Global Color Table を持っているとき true になります。

ColorResolution	number	R	元のイメージが持っている色のビット数から 1 を引いた数値が入ります。
SortFlag	boolean	R	Global Color Table のエントリが重要な色から順にソートされているとき true になります。
SizeOfGlobalColorTable	number	R	GlobalColorTableFlag が true のとき Global Color Table のバイト数を表します。
BackgroundColorIndex	number	R	Global Color Table 中の背景色のインデクス番号です。GlobalColorTableFlag が false のときは意味を持ちません。
BackgroundColor	color	R	Global Color Table 中の背景色を color インスタンスで取得します。 GlobalColorTableFlag が false のときは nil が返ります。
PixelAspectRatio	number	R	元の画像の縦横比を表します。 PixelAspectRatio が 0 のときは縦横比に関する情報はありません。それ以外のとき、縦横比は以下の式で表わされます。 (PixelAspectRatio + 15) / 64

プロパティ gif.content#CommentExtension は gif.CommentExtension クラスのインスタンスで、GIF フォーマット中の Comment Extension の内容を表します。以下のプロパティを持っています。

プロパティ	データ型	R/W	内容
CommentData	binary	R	コメントデータ

プロパティ gif.content#PlainTextExtension は gif.PlainTextExtension クラスのインスタンスで、GIF フォーマット中の Plain Text Extension の内容を表します。以下のプロパティを持っています。

プロパティ	データ型	R/W	内容
TextGridLeftPosition	number	R	テキストグリッドの左端の位置をピクセルで表わします
TextGridTopPosition	number	R	テキストグリッドの上端の位置をピクセルで表わします
TextGridWidth	number	R	テキストグリッドの幅をピクセルで表わします
TextGridHeight	number	R	テキストグリッドの高さをピクセルで表わします
CharacterCellWidth	number	R	グリッド内の各セルの幅をピクセルで表わします
CharacterCellHeight	number	R	グリッド内の各セルの高さをピクセルで表わします

TextForegroundColorIndex	number	R	テキスト前景色の Global Color Table のインデクス番号です。
TextBackgroundColorIndex	number	R	テキスト背景色の Global Color Table のインデクス番号です。
PlainTextData	binary	R	テキストデータ

プロパティ gif.content#ApplicationExtension は gif.ApplicationExtension クラスのインスタンスで、GIF フォーマット中の Application Extension の内容を表します。以下のプロパティを持っています。

プロパティ	データ型	R/W	内容
ApplicationIdentifier	binary	R	アプリケーションデータの識別子
AuthenticationCode	binary	R	ApplicationIdentifier を認証する 3 バイトデータです
ApplicationData	binary	R	アプリケーションデータ

15.4. image クラスの拡張

15.4.1. インスタンスメソッド

gif モジュールをインポートすることで以下のメソッドが image クラスに追加されます。

image#gifread(stream:stream):reduce

指定のストリームから GIF フォーマットのデータを読み込んで image インスタンスにデータを展開します。複数のイメージがある場合は、最初のイメージを読み込みます。

image#gifwrite(stream:stream):reduce

image インスタンスのデータを GIF フォーマットにして指定のストリームに書き込みます。このメソッドでは、複数のイメージを含む GIF ファイルは作成できません。

15.4.2. インスタンスプロパティ

GIF ファイルから、関数 image を使って image インスタンスを生成したり、image#gifread を使って内容を更新すると、image インスタンス内に gif という名前のプロパティが追加されます。プロパティ gif は gif.imgprop クラスのインスタンスで、以下のプロパティを持っています。

プロパティ	データ型	R/W	内容
GraphicControl	gif.GraphicControl	R	Graphic Control Extension
ImageDescriptor	gif.ImageDescriptor	R	Image Descriptor

15.4.3. インスタンスプロパティの詳細

image#gif.GraphicControl は gif.GraphicControl クラスのインスタンスで、GIF ファイル中、Image Descriptor に先行して表れた Graphic Control Extension の内容を表します。以下のプロパティを持っています。

プロパティ	データ型	R/W	内容
DisposalMethod	symbol	R	イメージを表示した後の処理を表します `none　なにもしません `keep　イメージを破棄しません `background 背景色に戻します `previous　前のイメージに戻します
UserInputFlag	boolean	R	イメージ処理を続ける前にユーザ入力が期待されているか否かを表します false　ユーザ入力なし true　ユーザ入力あり
TransparentColorFlag	boolean	R	背景色を有効にするか否かを表します false　背景色なし true　背景色あり
DelayTime	number	R	0 でない場合、次のイメージを処理するまでの 1/100 秒の遅延を表します。
TransparentColorIndex	number	R	背景色のインデクス値です。 TransparentColorFlag が true のとき有効です。

image#gif.ImageDescriptor は gif.ImageDescriptor クラスのインスタンスで、GIF フォーマット中の Image Descriptor の内容を表します。以下のプロパティを持っています。

プロパティ	データ型	R/W	内容
ImageLeftPosition	number	R	イメージの左端の位置をピクセルで表わします
ImageTopPosition	number	R	イメージの上端の位置をピクセルで表わします
ImageWidth	number	R	イメージの幅をピクセルで表わします
ImageHeight	number	R	イメージの高さをピクセルで表わします
LocalColorTableFlag	boolean	R	この値が true のときイメージは Local Color Table を持ちます
InterlaceFlag	boolean	R	true のとき、イメージがインターレースされていることを表します。
SortFlag	boolean	R	Local Color Table のエントリが重要な色から順にソートされているとき true になります。
SizeOfLocalColorTable	number	R	LocalColorTableFlag が true のとき Local Color Table のバイト数を表します。

15.4.4. パレットの扱い

GIF データの読み込み時、Image Descriptor が Local Color Table を持っている場合、その内容をエントリに持った palette インスタンスをイメージに登録します。Local Color Table が無い場合は Global Color Table の内容を入れた palette インスタンスを登録します。

GIF データを書き込む際、イメージが palette インスタンスを持っている場合はそのパレットを Global Color Table に書き込みます。palette インスタンスがない場合は Web-safe な色を持つパレットを作成して使用します。

16. jpeg モジュール

16.1. 概要

イメージデータを JPEG (Joint Photographic Experts Group) イメージフォーマットで読み書きするモジュールです。使用するには import 関数を使って jpeg モジュールをインポートします。

以下の URL で公開されている libjpeg ライブラリを内部で使用しています。

http://www.ijg.org/

16.2. ストリームの読み書き

image 関数で指定したストリームが以下のいずれかの条件に合致すると、それを JPEG イメージデータと認識して読み込み、image インスタンスを生成します。

- ストリームの識別子にサフィックス .jpeg、.jpg または .jpe がついている (大小文字の区別はなし)
- ストリームの先頭が 0xff, 0xd8 で始まっている (JPEG ストリームにおける Start of Image のシーケンス)

image#write メソッドで指定したストリームが以下の条件に合致すると、JPEG データを出力します。

- ストリームの識別子にサフィックス .jpeg、 .jpg または .jpe がついている (大小文字の区別はなし)

16.3. jpeg.exif クラス

16.3.1. 概要

JPEG ストリーム内の Exif フォーマットデータを扱うクラスです。

16.3.2. インスタンスの生成

```
jpeg.exif(stream?:stream:r):map:[raise] {block?}
```

jpeg.exif インスタンスを生成します。引数 stream を指定すると、そのストリームから Exif 形式のデータを読み込みます。ストリームが JPEG フォーマットとして認識できない場合はエラーになります。JPEG フォーマットになっているが、Exif のマーカーがない場合は nil を返します。アトリビュート :raise を指定すると、Exif マーカーがない場合エラーを通知します。

16.3.3. インスタンスプロパティ

プロパティ	データ型	R/W	内容
endian	symbol	R	Exif 内のエンディアンタイプがビッグエンディアンの場合`big、リトルエンディアンの場合`little を返します。
ifd0	jpeg.ifd	R	IFD0 の内容を返します。
ifd1	jpeg.ifd	R	IFD1 の内容を返します。

thumbnail	image	R	サムネールイメージデータを返します。サムネールイメージがない場合、nil を返します。
thumbnail_jpeg	binary	R	サムネールイメージの JPEG データを返します。サムネールイメージが存在していないか、サムネールが JPEG 形式以外の場合は nil を返します。

16.3.4. インスタンスメソッド

jpeg.exif#each() {block?}

IFD0 内に定義されているタグデータを要素に持つイテレータを返します。ifd0 プロパティに対して each()メソッドを実行した場合と同じです。

16.4. jpeg.ifd クラス

16.4.1. 概要

16.4.2. インスタンスの生成

16.4.3. インスタンスプロパティ

16.4.4. インスタンスメソッド

jpeg.ifd#each() {block?}

この IFD 内に定義されているタグデータを要素に持つイテレータを返します。

16.5. jpeg.tag クラス

16.5.1. 概要

16.5.2. インスタンスの生成

16.5.3. インスタンスプロパティ

プロパティ	データ型	R/W	内容
id	number	R	
name	string	R	
symbol	symbol	R	
type	number	R	

value	any	R	
ifd	jpeg.ifd	R	

16.6. image クラスの拡張

16.6.1. インスタンスメソッド

image#jpegread(stream:stream:r):reduce

指定のストリームから JPEG フォーマットのデータを読み込んで image インスタンスにデータを展開します。

image#jpegwrite(stream:stream:w):reduce

image インスタンスのデータを JPEG フォーマットにして指定のストリームに書き込みます。

17. msico モジュール

17.1. 概要

イメージデータを Microsoft アイコンファイルのフォーマットで読み書きするモジュールです。使用するには import 関数を使って msico モジュールをインポートします。

アイコンファイルは通常サイズの異なる複数のイメージを格納していますが、msico モジュールの content クラスを使うと、格納されたイメージを取得したり、新たなイメージを追加したりすることができます。

モジュールの実装は以下の URL の記述に基づきます。

http://msdn.microsoft.com/en-us/library/ms997538.aspx

17.2. ストリームの読み書き

image 関数で指定したストリームが以下のいずれかの条件に合致すると、それを ICO イメージデータと認識して読み込み、image インスタンスを生成します。

- ストリームの識別子にサフィックス .ico がついている（大小文字の区別はなし）

image#write メソッドで指定したストリームが以下の条件に合致すると、ICO イメージデータを出力します。

- ストリームの識別子にサフィックス .ico がついている（大小文字の区別はなし）

17.3. msico.content クラス

17.3.1. 概要

ICO ファイルは複数のイメージデータを格納できるフォーマットなので、単一の image インスタンスではファイル全体のデータ構造を処理することができません。msico.content クラスを使うと、複数のイメージデータを格納・参照することができるようになります。

17.3.2. インスタンスの生成

```
msico.content(stream?:stream:r, format:symbol => `rgba) {block?}
```

msico インスタンスを生成します。引数 stream を指定すると、そのストリームから ICO ファイル形式のデータを読み込みます。引数 format は、内部に保持する image インスタンスのフォーマットを指定します。

17.3.3. インスタンスメソッド

```
msico.content#addimage(image:image):map:reduce
```

msico インスタンスにイメージデータを追加します。

```
msico.content#write(stream:stream:w):reduce
```

msico インスタンスの内容を ICO ファイル形式でストリームに書き込みます。

17.4. image クラスの拡張

17.4.1. インスタンスメソッド

`image#msicoread(stream:stream:r):reduce`

ICO ファイル形式でストリームを読み込み、image インスタンスに展開します。複数のイメージが存在する場合は、最初のイメージを読み込みます。

`image#msicowrite(stream:stream:w):reduce`

image インスタンスの内容を ICO ファイル形式でストリームに書き込みます。このメソッドでは、複数のイメージを含む ICO ファイルは作成できません。

18. png モジュール

18.1. 概要

イメージデータを PNG (Portable Network Graphics) イメージフォーマットで読み書きするモジュールです。使用するには import 関数を使って png モジュールをインポートします。

以下の URL で公開されている libpng ライブラリを内部で使用しています。

http://www.libpng.org/pub/png/libpng.html

18.2. ストリームの読み書き

image 関数で指定したストリームが以下のいずれかの条件に合致すると、それを PNG イメージデータと認識して読み込み、image インスタンスを生成します。

- ストリームの識別子にサフィックス .png がついている (大小文字の区別はなし)
- ストリームの先頭が 0x89, 0x50, 0x4e, 0x47, 0x0d, 0x0a, 0x1a, 0x0a で始まっている

image#write メソッドで指定したストリームが以下の条件に合致すると、PNG イメージデータを出力します。

- ストリームの識別子にサフィックス .png がついている (大小文字の区別はなし)

18.3. image クラスの拡張

18.3.1. インスタンスメソッド

image#pngread(stream:stream:r):reduce

指定のストリームから PNG フォーマットのデータを読み込んで image インスタンスにデータを展開します。

image#pngwrite(stream:stream:w):reduce

image インスタンスのデータを PNG フォーマットにして指定のストリームに書き込みます。

19. ppm モジュール

19.1. 概要

イメージデータを PPM (Portable Pixmap) イメージフォーマットで読み書きするモジュールです。使用するには import 関数を使って ppm モジュールをインポートします。

モジュールの実装は以下の URL の記述に基づきます。

http://local.wasp.uwa.edu.au/~pbourke/dataformats/ppm/

19.2. ストリームの読み書き

image 関数で指定したストリームが以下のいずれかの条件に合致すると、それを PPM イメージデータと認識して読み込み、image インスタンスを生成します。

- ストリームの識別子にサフィックス .ppm または .pbm がついている (大小文字の区別はなし)
- ストリームの先頭が "P2"、"P3" または "P6" で始まっている

image#write メソッドで指定したストリームが以下の条件に合致すると、PPM イメージデータを出力します。

- ストリームの識別子にサフィックス .ppm または .pbm がついている (大小文字の区別はなし)

19.3. image クラスの拡張

19.3.1. インスタンスメソッド

image#ppmread(stream:stream:r):reduce

指定のストリームから PPM フォーマットのデータを読み込んで image インスタンスにデータを展開します。

image#ppmwrite(stream:stream:w):reduce

image インスタンスのデータを PPM フォーマットにして指定のストリームに書き込みます。

20. xpm モジュール

20.1. 概要

イメージデータを XPM (X Pixmap) イメージフォーマットで出力するモジュールです。使用するには import 関数を使って xpm モジュールをインポートします。

20.2. ストリームの書きこみ

image#write メソッドで指定したストリームが以下の条件に合致すると、XPM イメージデータを出力します。

- ストリームの識別子にサフィックス .xpm がついている (大小文字の区別はなし)

20.3. image クラスの拡張

20.3.1. インスタンスメソッド

image#xpmwrite(stream:stream:w):reduce

image インスタンスのデータを XPM フォーマットにして指定のストリームに書き込みます。

21. freetype モジュール

21.1. 概要

image インスタンスにテキストの描画を行います。使用するには import 関数を使って freetype モジュールをインポートします。

以下の URL で公開されている FreeType ライブラリを内部で使用しています。

http://www.freetype.org/

21.2. 関数

freetype.sysfontpath(name?:string):map

システムフォントが格納されているディレクトリパスを返します。引数 name を指定すると、ディレクトリとその名前を結合した結果を返します。

21.3. freetype.font クラス

21.3.1. 概要

freetype.font クラスは、回転や斜体表示などの修飾に必要な属性値を管理し、フォントの描画処理を行います。

21.3.2. インスタンスの生成

freetype.font(face:freetype.Face):map

freetype.Face クラスのインスタンスを持った feetype.font インスタンスを生成します。

21.3.3. インスタンスメソッド

freetype.font#calcbbox(x:number, y:number, str:string):map

t.b.d

freetype.font#calcsize(str:string):map

文字列 str を描画したときのサイズを[width, height]という形式で返します。

freetype.font#cleardeco():reduce

修飾要素をすべてとりのぞきます。

freetype.font#drawtext(image:image, x:number, y:number, str:string):map:reduce

image インスタンスの指定の位置に文字列を描画します。

21.3.4. インスタンスプロパティ

プロパティ	データ型	R/W	説明
mode	symbol	R/W	`blend を指定すると、フォントをイメージ上に描画する

			際、諧調情報をもとにイメージの元の色とブレンド処理を行います。`alpha を指定すると、諧調情報はアルファ値としてイメージに書き込まれます。
color	color	R/W	描画時の色を指定します。
width	number	R/W	フォントの幅をピクセル値で指定します。
height	number	R/W	フォントの高さをピクセル値で指定します。
slant	number	R/W	フォントの傾きを設定します。0 のとき傾きなし、1 で文字を 45 度傾けます。
strength	number	R/W	フォントの太さを設定します。0 でノーマル、1 で約二倍、2 で約三倍の太さになります。
rotate	number	R/W	文字列の左下を原点にして文字列を、degree 度だけ回転します。degree が正の値のとき反時計まわりに回転します。
face	freetype.Face	R/W	freetype.Face クラスのインスタンスを返します。

21.4. freetype.Face クラス

21.4.1. インスタンスの生成

```
freetype.Face(stream:stream, index:number => 0):map
```

指定のストリームからフォントデータを読み込み、index 番目のフォントをもとに feetype.Face インスタンスを生成します。

21.4.2. インスタンスプロパティ

プロパティ	データ型	R/W	説明
num_faces	number	R	
face_index	number	R	
family_name	string	R	
style_name	string	R	
bbox	list	R	
ascender	number	R	
descender	number	R	
height	number	R	
max_advance_width	number	R	
max_advance_height	number	R	
underline_position	number		
underline_thickness	number		
glyph	freetype.GlyphSlot		
size			
charmap			

21.4.3. インスタンスメソッド

```
freetype.Face#CheckTrueTypePatents()
```

freetype.Face インスタンスに対して FT_Face_CheckTrueTypePatents 関数を実行します。

```
freetype.Face#Get_Advance(glyph_index:number, load_flags:number)
```

freetype.Face インスタンスに対して FT_Get_Advance 関数を実行します。

```
freetype.Face#Get_Advances(glyph_index_start:number,
                                    count:number, load_flags:number)
```

freetype.Face インスタンスに対して FT_Get_Advances 関数を実行します。

```
freetype.Face#Get_Glyph_Name(glyph_index:number)
```

freetype.Face インスタンスに対して FT_Get_Glyph_Name 関数を実行します。

```
freetype.Face#Get_Postscript_Name()
```

freetype.Face インスタンスに対して FT_Get_Postscript_Name 関数を実行します。

```
freetype.Face#Get_Kerning()
```

freetype.Face インスタンスに対して FT_Get_Kerning 関数を実行します。

```
freetype.Face#Load_Char(char_code:number, load_flags:number)
```

freetype.Face インスタンスに対して FT_Load_Char 関数を実行します。

```
freetype.Face#Load_Glyph(glyph_index:number, load_flags:number)
```

freetype.Face インスタンスに対して FT_Load_Glyph 関数を実行します。

```
freetype.Face#Set_Charmap(charmap:freetype.CharMap):reduce
```

freetype.Face インスタンスに対して FT_Set_Charmap 関数を実行します。

```
freetype.Face#Set_Pixel_Sizes(pixel_width:number, pixel_height:number):reduce
```

freetype.Face インスタンスに対して FT_Set_Pixel_Sizes 関数を実行します。

21.5. freetype.GlyphSlot クラス

21.5.1. インスタンスプロパティ

プロパティ	データ型	R/W	説明
advance	freetype.Vector	R	
format	number	R	
bitmap	freetype.Bitmap		
bitmap_left	number		
bitmap_top	number		
outline	freetype.Outline		

21.5.2. インスタンスメソッド

`freetype.GlyphSlot#Get_Glyph()`

`freetype.Glyph` インスタンスを返します。

`freetype.GlyphSlot#Render(render_mode:number)`

`GlyphSlot` 内のビットマップに対して描画処理を行います。

21.6. `freetype.Outline` クラス

`freetype.Outline#Translate(xOffset:number, yOffset:number):reduce`

`freetype.Outline` インスタンスに対して `FT_Outline_Translate` 関数を実行します。

`freetype.Outline#Transform(matrix:freetype.Matrix):reduce`

`freetype.Outline` インスタンスに対して `FT_Outline_Transform` 関数を実行します。

`freetype.Outline#Embolden(strength:number):reduce`

`freetype.Outline` インスタンスに対して `FT_Outline_Embolden` 関数を実行します。

`freetype.Outline#EmboldenXY():reduce`

`freetype.Outline` インスタンスに対して `FT_Outline_EmboldenXY` 関数を実行します。

`freetype.Outline#Reverse():reduce`

`freetype.Outline` インスタンスに対して `FT_Outline_Reverse` 関数を実行します。

21.7. `freetype.Glyph` クラス

21.8. `freetype.Matrix` クラス

21.8.1. インスタンスの生成

`freetype.Matrix(matrix:matrix):map {block?}`

Gura の組み込みクラス `matrix` のインスタンスから `freetype.Matrix` インスタンスを生成します。もとのマトリクスは 2 行 2 列の正方行列でなければいけません。

21.8.2. インスタンスメソッド

`freetype.Matrix#Multiply(matrix:freetype.Matrix):reduce`

`freetype.Matrix` インスタンスに別の行列をかけあわせます。

`freetype.Matrix#Invert():reduce`

`freetype.Matrix` インスタンスを逆行列にします。

21.9. freetype.Vector クラス

21.9.1. インスタンスの生成

freetype.Vector(x:number, y:number):map {block?}

feetype.Vector インスタンスを生成します。

21.10. image クラスの拡張

21.10.1. インスタンスメソッド

image#drawtext(font:freetype.font, x:number, y:number, str:string):map:reduce

イメージの指定の位置に文字列を描画します。

22. sqlite3 モジュール

22.1. 概要

SQLite3 のデータベースにアクセスするためのモジュールです。使用するには import 関数を使って sqlite3 モジュールをインポートします。

以下の URL で公開されている SQLite3 クライアントライブラリを内部で使用しています。

http://www.sqlite.org

22.2. データオブジェクトの対応

SQLite3 のデータ型とスクリプトのデータ型は以下のように対応しています。

SQLite3	スクリプト
SQLITE_INTEGER	number
SQLITE_FLOAT	number
SQLITE_TEXT	string
SQLITE_BLOB	(未対応)
SQLITE_NULL	nil

22.3. sqlite3.db クラス

22.3.1. インスタンスの生成

sqlite3.db(filename:string) {block?}

データベースファイルを指定し、sqlite3.db インスタンスを生成します。

22.3.2. インスタンスメソッド

sqlite3.db#close()

データベースをクローズします。

sqlite3.db#exec(sql:string):map

SQL 文を実行します。

sqlite3.db#getcolnames(sql:string):map {block?}

SQL 文を実行した結果のカラム名をリストにして返します。

sqlite3.db#query(sql:string):map {block?}

SQL 文を実行した結果を返すイテレータを生成します。

sqlite3.db#transaction() {block}

SQLite3 コマンド "BEGIN TRANSACTION" を実行して block を評価し、その後 SQLite3 コマンド"END TRANSACTION" を実行します。

23. gzip モジュール

23.1. 概要

gzip 形式によるストリームデータの圧縮および展開を行います。使用するには import 関数を使って gzip モジュールをインポートします。

以下の URL で公開されている zlib ライブラリを内部で使用しています。

http://zlib.net/

23.2. モジュール変数

圧縮レベルを表す以下の数値が変数に定義されています。

変数	型	内容
NO_COMPRESSION	number	圧縮なし (0)
BEST_SPEED	number	最も速度効率が高い (1)
BEST_COMPRESSION	number	最も高い圧縮率 (9)
DEFAULT_COMPRESSION	number	デフォルト (-1)

23.3. モジュール関数

```
gzip.reader(stream:stream:r) {block?}
```

ストリーム stream から gzip 形式で圧縮されたデータ列を読み込み、展開した結果を返すストリームを返します。

```
gzip.writer(stream:stream:w, level?:number) {block?}
```

ストリーム stream に gzip 形式で圧縮したデータ列を書きこむストリームを返します。

引数 level に 0 から 9 まで数値で圧縮レベルを指定します。0 が圧縮なし、9 が最も圧縮率が高い設定になります。

23.4. stream クラスの拡張

23.4.1. インスタンスメソッド

```
stream#gzipreader()
```

ストリーム stream から gzip 形式で圧縮されたデータ列を読み込み、展開した結果を返すストリームを返します。

```
stream#gzipwriter(level?:number)
```

ストリーム stream に gzip 形式で圧縮したデータ列を書きこむストリームを返します。

引数 level に 0 から 9 まで数値で圧縮レベルを指定します。0 が圧縮なし、9 が最も圧縮率が高い設定になります。

24. bzip2 モジュール

24.1. 概要

bzip2 形式によるストリームデータの圧縮および展開を行います。使用するには import 関数を使って bzip2 モジュールをインポートします。

以下の URL で公開されている libbz2 ライブラリを内部で使用しています。

http://www.bzip.org/

24.2. モジュール関数

bzip2.reader(stream:stream:r) {block?}

ストリーム stream から bzip2 形式で圧縮されたデータ列を読み込み、展開した結果を返すストリームを返します。

bzip2.writer(stream:stream:w) {block?}

ストリーム stream に bzip2 形式で圧縮したデータ列を書きこむストリームを返します。

24.3. stream クラスの拡張

24.3.1. インスタンスメソッド

stream#bzip2reader() {block?}

ストリーム stream から bzip2 形式で圧縮されたデータ列を読み込み、展開した結果を返すストリームを返します。

stream#bzip2writer() {block?}

ストリーム stream に bzip2 形式で圧縮したデータ列を書きこむストリームを返します。

25. zip モジュール

25.1. 概要

ZIP アーカイブの操作をするモジュールです。使用するには import 関数を使って zip モジュールをインポートします。

以下の URL で公開されているライブラリを内部で使用しています。

http://zlib.net/	zlib
http://www.bzip.org/	libbz2

25.2. パス名の拡張

パス名の途中にサフィックス .zip がついた要素が存在し、それがファイルであれば、その要素以下のパスで指定されるディレクトリやファイルは zip モジュールによって処理されます。

この拡張により、以下の操作が可能になります。

- open 関数で ZIP アーカイブ中のファイルをオープンできるようになります。
- ストリームを受け取る引数に、ZIP アーカイブ中のファイルパス名を指定できるようになります。
- path.dir, path.walk, path.glob 関数で、ZIP アーカイブ中のディレクトリパスをサーチできるようになります。

25.3. zip.reader クラス

25.3.1. インスタンスの生成

```
zip.reader(stream:stream:r) {block?}
```

ストリームから ZIP アーカイブデータを読み込む zip.reader インスタンスを生成します。

25.3.2. インスタンスメソッド

```
zip.reader#entries() {block?}
```

ZIP アーカイブ中のファイルを読み取るストリームを返すイテレータを生成します。

25.4. zip.writer クラス

25.4.1. インスタンスの生成

```
zip.writer(stream:stream:w, compression?:symbol) {block?}
```

ストリームに ZIP アーカイブデータを書き込む zip.writer インスタンスを生成します。引数 compression には圧縮形式を以下のシンボルから指定します。

- `store　非圧縮
- `deflate　gzip 形式による圧縮（デフォルト）
- `bzip2　bzip2 形式による圧縮

25.4.2. インスタンスメソッド

```
zip.writer#add(stream:stream:r, filename?:string,
                                    compression?:symbol):map:reduce
```

ストリームの内容をもったエントリを ZIP アーカイブに追加します。引数 `filename` をつけるとその名前でエントリを作成します。省略した場合、ストリームの名前がエントリにつけられます。

`compression` にはこのエントリに対する圧縮形式を `zip.writer` 関数と同じシンボルで指定します。省略した場合、`zip.writer` で指定した `compression` を適用します。

```
zip.writer#close():reduce
```

Central Directory Record の追加やストリームのフラッシュなど、必要な後処理を行います。

25.5. zip.stat クラス

25.5.1. インスタンスプロパティ

メソッド `zip.reader#entries` で返すストリームには、`stat` という名前のプロパティがあり `zip.stat` 型のインスタンスです。このインスタンスは以下のプロパティを持ちます。

プロパティ	データ型	R/W	内容
filename	string	R	ファイル名
comment	string	R	コメント
mtime	datetime	R	最終更新日時
crc32	number	R	CRC32 チェックサム
compression_method	number	R	圧縮形式
size	number	R	圧縮前のサイズ
compressed_size	number	R	圧縮後のサイズ
attributes	number	R	アトリビュート

26. tar モジュール

26.1. 概要

TAR アーカイブの操作をするモジュールです。使用するには import 関数を使って tar モジュールをインポートします。

通常の TAR ファイルに加え、gzip で圧縮された TAR ファイル（サフィックス .tgz または.tar.gz）および bzip2 で圧縮された TAR ファイル（サフィックス .tar.bz2）も処理できます。

モジュールの実装は以下の URL の記述に基づきます。

http://www.gnu.org/software/tar/manual/html_node/Standard.html

以下の URL で公開されているライブラリを内部で使用しています。

http://zlib.net/	zlib
http://www.bzip.org/	libbz2

なお、tar モジュールは zlib や libbz2 ライブラリを内包しているので、gzip や bzip2 モジュールをインポートする必要はありません。

26.2. パス名の拡張

パス名の途中に以下のいずれかのサフィックスがついた要素名が存在し、それがファイルであれば、その要素以下のパスで指定されるディレクトリやファイルは tar モジュールによって処理されます。

.tar .tar.gz .tgz .tar.bz2

この拡張により、以下の操作が可能になります。

- open 関数で TAR アーカイブ中のファイルをオープンできるようになります。
- ストリームを受け取る引数に、TAR アーカイブ中のファイルパス名を指定できるようになります。
- path.dir, path.walk, path.glob 関数で、TAR アーカイブ中のディレクトリパスをサーチできるようになります。

26.3. モジュール変数

ファイルタイプを表す以下の値が変数に割り当てられています。tar.stat クラスのプロパティ typeflag で参照されます。

変数	型	内容
REGTYPE	number	regular file
AREGTYPE	number	regular file
LNKTYPE	number	link
SYMTYPE	number	(reserved)
CHRTYPE	number	character special

BLKTYPE	number	block special
DIRTYPE	number	directory
FIFOTYPE	number	FIFO special
CONTTYPE	number	(reserved)
XHDTYPE	number	Extended header referring to the next file in the archive
XGLTYPE	number	Global extended header

26.4. tar.reader クラス

26.4.1. インスタンスの生成

```
tar.reader(stream:stream:r, compression?:symbol) {block?}
```

ストリームから TAR アーカイブデータを読み込む tar.reader インスタンスを生成します。

引数 compression を省略すると、ストリームの名前がサフィックス .tar.gz または .tgz を持っているとき gzip 展開して読み込みます。また、ストリームの名前がサフィックス .tar.bz2 を持っているとき bzip2 展開して読み込みます。

compression に以下の symbol を指定してストリームデータの展開方法を指定することができます。

- `none　展開処理なし
- `auto　ストリームの名前のサフィックスによる自動認識 (デフォルト)
- `gzip　gzip 展開
- `bzip2　bzip2 展開

26.4.2. インスタンスメソッド

```
tar.reader#entries() {block?}
```

TAR アーカイブ中のファイルを読み取るストリームを返すイテレータを生成します。

26.5. tar.writer クラス

26.5.1. インスタンスの生成

```
tar.writer(stream:stream:w, compression?:symbol) {block?}
```

ストリームに TAR アーカイブデータを書き込む tar.writer インスタンスを生成します。

引数 compression を省略すると、ストリームの名前がサフィックス.tar.gz または.tgz を持っているとき gzip 圧縮して書き込みます。また、ストリームの名前がサフィックス.tar.bz2 を持っているとき bzip2 圧縮して書き込みます。

compression に以下の symbol を指定してストリームデータの圧縮方法を指定することができます。

- `none　圧縮処理なし
- `auto　ストリームの名前のサフィックスによる自動認識 (デフォルト)
- `gzip　gzip 圧縮
- `bzip2　bzip2 圧縮

26.5.2. インスタンスメソッド

`tar.writer#add(stream:stream:r, filename?:string):map:reduce`

ストリームの内容をもったエントリを TAR アーカイブに追加します。引数 `filename` をつけるとその名前でエントリを作成します。省略した場合、ストリームの名前がエントリにつけられます。

`tar.writer#close():reduce`

ターミネータブロックの追加やストリームのフラッシュなど、必要な後処理を行います。

26.6. `tar.stat` クラス

26.6.1. インスタンスプロパティ

メソッド `tar.reader#entries` で返すストリームには、`stat` という名前の `tar.stat` 型のプロパティがあります。このプロパティは以下のメンバを持ちます。

プロパティ	データ型	R/W	説明
`name`	`string`	R	ファイル名
`linkname`	`string`	R	リンク名
`uname`	`string`	R	ユーザ名
`gname`	`string`	R	グループ名
`mode`	`number`	R	アクセスモード
`uid`	`number`	R	ユーザ ID
`gid`	`number`	R	グループ ID
`size`	`number`	R	ファイルサイズ
`mtime`	`datetime`	R	更新日時
`atime`	`datetime`	R	アクセス日時
`ctime`	`datetime`	R	作成日時
`chksum`	`number`	R	ヘッダブロック内のチェックサム
`typeflag`	`number`	R	ファイルタイプ
`devmajor`	`number`	R	デバイスメジャー番号
`devminor`	`number`	R	デバイスマイナー番号

27. curl モジュール

27.1. 概要

さまざまなプロトコルを用いてデータ転送を行う cURL ライブラリを操作するモジュールです。使用するには import 関数を使って curl モジュールをインポートします。

以下の URL で公開されているライブラリを内部で使用しています。

http://curl.haxx.se/ libcurl

27.2. パス名の拡張

パス名が "http:"、"https:"、"ftp:"、"ftps:"、"sftp:" のいずれかで始まっていると、curl モジュールによってパスやストリームを処理します。

この拡張により、以下の操作が可能になります。

- open 関数で HTTP プロトコルや FTP プロトコルを通じたファイルをオープンできるようになります。
- ストリームを受け取る引数に、HTTP や FTP のファイルパス名を指定できるようになります。

27.3. モジュール関数

```
curl.version()
```

cURL のバージョン文字列を返します。

27.4. curl.easy_handle クラス

27.4.1. インスタンスの生成

```
curl.easy_init() {block?}
```

cURL のハンドルを内包した curl.easy_handle インスタンスを生成します。

27.4.2. インスタンスメソッド

```
curl.easy_handle#escape(string:string):void
```

```
curl.easy_handle#getinfo(info:number)
```

```
curl.easy_handle#pause(bitmask:number):void
```

```
curl.easy_handle#perform(stream?:stream:w):void
```

```
curl.easy_handle#recv(buflen:number)
```

```
curl.easy_handle#reset():void
```

```
curl.easy_handle#send(buffer:binary)
```

```
curl.easy_handle#setopt(option:number, arg):void
```

```
curl.easy_handle#unescape(string:string):void
```

28. re モジュール

28.1. 概要

正規表現を処理するモジュールです。使用するには import 関数を使って re モジュールをインポートします。

以下の URL で公開されている鬼車ライブラリを内部で使用しています。

http://www.geocities.jp/kosako3/oniguruma/

正規表現を用いて、以下の文字列操作をすることができます。

- パターンマッチング – match
- 連続パターンマッチング – scan
- 文字列分割 – split
- 置換 – sub

28.2. 正規表現パターン記述について

re モジュールでは、re.pattern インスタンスで正規表現パターンを扱います。
正規表現の文法は POSIX の拡張正規表現に従います。将来的に正規表現のエンジンを変更する可能性もあるので、鬼車ライブラリで独自拡張されたパターンの使用は推奨しません。
re.pattern を引数にとる関数またはメソッドに文字列を指定すると、型キャストにより自動的に re.pattern インスタンスを生成し、引数として渡します。以下の二つの記述は等価です。

```
re.match(r'¥w+', str)
re.match(re.pattern(r'¥w+'), str)
```

正規表現のパターン文字列中では、文字種を指定したり正規表現記号を無効化したりするためにバッククオート記号 "¥" を多用します。そのため、正規表現パターンを記述するには、上記のように "r" プレフィックスつきで文字列を作成し、バッククオートをスクリプトのパーサに通常文字として認識させるようにしておくと便利です。

re.pattern インスタンスはパターン文字列をもとに正規表現のパーサをコンパイルして生成されます。このため、大量の文字列を扱うとき、re.pattern インスタンスの生成を伴う文字列からのキャストがパフォーマンスを低下させる原因になります。あらかじめ re.pattern インスタンスを生成しておき、これを引数として渡すことで、評価効率を上げることができます。

28.3. モジュール関数

re.match(pattern:re.pattern, str:string, pos:number => 0, endpos?:number):map

正規表現 pattern に文字列 str がマッチしたとき re.match インスタンスを返します。マッチしない場合は nil を返します。
引数 pos と endpos でパターンをマッチさせる範囲を文字単位で指定することができます。endpos は範囲に含める文字の次の文字位置を指定します。これらを省略した場合、文字列全体が処理対象になりま

す。

re.scan(pattern:re.pattern, str:string, pos:number => 0, endpos?:number):map {block?}

正規表現 pattern に文字列 str がマッチしたとき re.match インスタンスを返すイテレータを生成します。マッチすると、マッチした文字列の次の文字からパターンマッチングを行います。このようにして、パターンがマッチしなくなるまで re.match インスタンスを返します。

引数 pos と endpos でパターンをマッチさせる範囲を文字単位で指定することができます。endpos は範囲に含める文字の次の文字位置を指定します。これらを省略した場合、文字列全体が処理対象になります。

re.split(pattern:re.pattern, str:string, count?:number):map {block?}

正規表現 pattern であらわされるパターンで文字列 str を分割します。引数 count を指定すると、分割数をその数までに限定します。

re.sub(pattern:re.pattern, replace, str:string, count?:number):map

文字列 str 中、正規表現 pattern にマッチする部分を replace で置換します。引数 count を指定すると、置換する回数をその数までに限定します。

replace には文字列または関数を指定します。

replace に文字列を指定したとき、マッチ部分をその文字列で置き換えます。このとき、replace 文字列中に "¥0" という記述があったとき、この部分をマッチした全体の文字列で置き換えます。また、"¥1"、"¥2" … という記述はそれぞれマッチパターンのグループ 1、グループ 2…の文字列で置き換えます。

replace に関数を渡したとき、関数を以下の形式で呼び出し、マッチ部分をこの関数の戻り値で置き換えます。

```
replace(strSub:string)
```

戻り値が文字列でない場合は、文字列に変換してから置き換えます。

28.4. re.match クラス

28.4.1. インスタンスの生成

re.match インスタンスは以下の関数またはメソッドで生成されます。

- モジュール関数 re.match
- re.pattern クラスのメソッド re.pattern#match
- string クラスに追加されるメソッド string#match

28.4.2. マッチパターンの取得

m が re.match のインスタンスであるとすると、m[0]を参照するとマッチした全体の文字列が返ります。また、m[1]は 1 番目のグループの文字列、m[2]は 2 番のグループ文字列と続きます。[] を用いたグループ文字列参照は、re.match#group メソッドを使った場合と等価です。

グループに名前がついているとき、インデクスに名前文字列を指定することができます。グループ名は、以下の例のようにグループの内部に "?<" と ">" で囲んで記述します。

```
m = re.pattern(r'(?<first>¥d+)¥.(?<second>¥d*)').match('3.14')
```

この例の場合、最初のグループは m['first']、二番目のグループは m['second'] というように参照できます。

28.4.3. インスタンスプロパティ

プロパティ	型	R/W	説明
string	string	R	比較した文字列全体を返します

28.4.4. インスタンスメソッド

re.match#end(index):map

引数 index で指定したグループの終了位置を返します。

re.match#group(index):map

引数 index で指定したグループの文字列を返します。この処理は、[] を用いたグループ参照と同じです。

re.match#groups()

マッチしたグループの文字列をリストにして返します。

re.match#start(index):map

引数 index で指定したグループの開始位置を返します。

28.5. re.pattern クラス

28.5.1. インスタンスの生成

re.pattern(pattern:string):map:[icase,multiline]

正規表現を記述した文字列から re.pattern インスタンスを返します。

アトリビュート:icase をつけると、マッチングの際大文字と小文字の区別をつけません。

アトリビュート:multiline をつけると、改行コードが含まれている文字列を処理できるようになります。

28.5.2. インスタンスメソッド

re.pattern#match(str:string, pos:number => 0, endpos?:number):map

正規表現 pattern に文字列 str がマッチしたとき re.match インスタンスを返します。マッチしない場合は nil を返します。

引数 pos と endpos でパターンをマッチさせる範囲を文字単位で指定することができます。endpos は範囲に含める文字の次の文字位置を指定します。これらを省略した場合、文字列全体が処理対象になります。

re.pattern#scan(str:string, pos:number => 0, endpos?:number):map {block?}

正規表現 pattern に文字列 str がマッチしたとき re.match インスタンスを返すイテレータを生成します。マッチすると、マッチした文字列の次の文字からパターンマッチングを行います。このようにして、パタ

ーンがマッチしなくなるまで re.match インスタンスを返します。

引数 pos と endpos でパターンをマッチさせる範囲を文字単位で指定することができます。endpos は範囲に含める文字の次の文字位置を指定します。これらを省略した場合、文字列全体が処理対象になります。

re.pattern#split(str:string, count?:number):map {block?}

正規表現 pattern であらわされるパターンで文字列 str を分割します。引数 count を指定すると、分割数をその数までに限定します。

re.pattern#sub(replace, str:string, count?:number):map

文字列 str 中、正規表現 pattern にマッチする部分を replace で置換します。引数 count を指定すると、置換する回数をその数までに限定します。

replace には文字列または関数を指定します。詳細についてはモジュール関数 re.sub の説明を参照ください。

28.6. string クラスの拡張

28.6.1. インスタンスメソッド

re モジュールをインポートすると、string クラスに以下のメソッドが追加されます。

string#match(pattern:re.pattern, pos:number => 0, endpos?:number):map

正規表現 pattern に string インスタンスがマッチしたとき re.match インスタンスを返します。マッチしない場合は nil を返します。

引数 pos と endpos でパターンをマッチさせる範囲を文字単位で指定することができます。endpos は範囲に含める文字の次の文字位置を指定します。これらを省略した場合、文字列全体が処理対象になります。

string#scan(pattern:re.pattern, pos:number => 0, endpos?:number):map {block?}

正規表現 pattern に string インスタンスがマッチしたとき re.match インスタンスを返すイテレータを生成します。マッチすると、マッチした文字列の次の文字からパターンマッチングを行います。このようにして、パターンがマッチしなくなるまで re.match インスタンスを返します。

引数 pos と endpos でパターンをマッチさせる範囲を文字単位で指定することができます。endpos は範囲に含める文字の次の文字位置を指定します。これらを省略した場合、文字列全体が処理対象になります。

string#splitreg(pattern:re.pattern, count?:number):map {block?}

正規表現 pattern であらわされるパターンで string インスタンスを分割します。引数 count を指定すると、分割数をその数までに限定します。

string#split メソッドは string クラスに元から備わっているインスタンスメソッドで、通常の文字列パターンによる文字列区切りを行います。

string#sub(pattern:re.pattern, replace, count?:number):map

string インスタンス中、正規表現 pattern にマッチする部分を replace で置換します。引数 count を指定すると、置換する回数をその数までに限定します。

replace には文字列または関数を指定します。詳細についてはモジュール関数 re.sub の説明を参照ください。

28.7. list/iterator クラスの拡張

28.7.1. インスタンスメソッド

re モジュールをインポートすると、list および iterator クラスに以下のメソッドが追加されます。

```
list#grep(pattern:re.pattern) {block?}
iterator#grep(pattern:re.pattern) {block?}
```

リストまたはイテレータの要素を文字列にして正規表現 pattern と比較し、マッチしたときの re.match インスタンスを要素にするイテレータを返します。

このメソッドは、メンバマッピングを使ってリストやイテレータに対して match メソッドを実行した後、skipnil で nil 要素を取り除く処理と同じです。リスト tbl があったとき、以下の二つの呼び出しは等価です。

```
tbl:*match(r'¥w+').skipnil()
tbl.grep(r'¥w+')
```

29. csv モジュール

29.1. 概要

CSV ファイルの読み書きを行います。使用するには import 関数を使って csv モジュールをインポートします。

実装は RFC4180 で記述される仕様に基づきます。

29.2. モジュール関数

```
csv.parse(str:string):map
```

CSV 形式のテキストを含んだ文字列を受け取り、カンマで区切られたフィールドの値を要素に持つリストを一行ずつ返すイテレータを生成します。

```
csv.reader(stream:stream:r) {block?}
```

ストリームから CSV 形式のテキストデータを読み込み、カンマで区切られたフィールドの値を要素に持つリストを一行ずつ返すイテレータを生成します。

```
csv.writer(stream:stream:w, format?:string) {block?}
```

ストリームに CSV 形式のテキストデータ出力する csv.writer インスタンスを生成します。

引数 format には数値データのフォーマット文字列を指定します。省略すると '%g' が使用されます。

29.3. csv.writer クラス

29.3.1. インスタンスプロパティ

プロパティ	データ型	R/W	説明
format	string	R/W	フォーマット文字列。書きかえることで数値データの出力フォーマットを途中で変えることができます。

29.3.2. インスタンスメソッド

```
csv.writer#write(fields+) {block?}
```

引数 fields に与えた値をカンマでつなげ、CSV 形式のテキストにして出力します。

29.4. stream クラスの拡張

29.4.1. インスタンスメソッド

```
stream#csvreader() {block?}
```

ストリームから CSV 形式のテキストデータを読み込み、カンマで区切られたフィールドの値を要素に持つリストを一行ずつ返すイテレータを生成します。

```
stream#csvwriter(format?:string) {block?}
```

ストリームに CSV 形式のテキストデータ出力する csv.writer インスタンスを生成します。

引数 format には数値データのフォーマット文字列を指定します。省略すると '%g' が使用されます。

30. xml モジュール

30.1. 概要

XML ファイルの読み書きを行います。使用するには import 関数を使って xml モジュールをインポートします。

以下の URL で公開されている Expat ライブラリを内部で使用しています。

http://expat.sourceforge.net/

30.2. モジュール関数

```
xml.read(stream:stream:r)
```

30.3. xml.parser クラス

30.3.1. インスタンスの生成

```
xml.parser() {block?}
```

30.3.2. オーバーライドメソッド

```
xml.parser#StartElement(element:xml.element)

xml.parser#EndElement(name:string)

xml.parser#CharacterData(text:string)

xml.parser#ProcessingInstruction(target:string, data:string)

xml.parser#Comment(data:string)

xml.parser#StartCdataSection()

xml.parser#EndCdataSection()

xml.parser#Default(text:string)

xml.parser#DefaultExpand(text:string)

xml.parser#ExternalEntityRef()
```

```
xml.parser#SkippedEntity(entityName:string, isParameterEntity:boolean)

xml.parser#StartNamespaceDecl(prefix:string, uri:string)

xml.parser#EndNamespaceDecl(prefix:string)

xml.parser#XmlDecl(version:string, encoding:string, standalone?:boolean)

xml.parser#StartDoctypeDecl(doctypeName:string, systemId:string,
    publicId:string, hasInternalSubset:boolean)

xml.parser#EndDoctypeDecl()

xml.parser#ElementDecl(name:string, type:symbol)

xml.parser#AttlistDecl(elemName:string, attName:string, attType:string,
    default:string, isRequired:boolean)

xml.parser#EntityDecl(entityName:string, isParameterEntity:boolean,
    value:string,base:string, systemId:string,
     publicId:string, notationName:string)

xml.parser#NotationDecl(notationName:string, base:string,
    systemId:string, publicId:string)

xml.parser#NotStandalone()
```

30.3.3. インスタンスプロパティ

プロパティ	データ型	R/W	説明
children	xml.element[]	R/W	

30.3.4. インスタンスメソッド

```
xml.parser#parse(stream:stream)
```

30.4. xml.element クラス

30.4.1. インスタンスの生成

```
xml.element(name:string, attrs%) {block?}
```

30.5. stream クラスの拡張

30.5.1. インスタンスメソッド

```
stream#xmlread()
```

31. yaml モジュール

31.1. 概要

YAML ファイルの読み書きを行います。使用するには import 関数を使って yaml モジュールをインポートします。

以下の URL で公開されている yaml ライブラリを内部で使用しています。

http://www.yaml.org/

31.2. データオブジェクトの対応

31.3. モジュール関数

yaml.compose(obj)

obj の内容を YAML フォーマットの文字列にします。

yaml.parse(str:string)

YAML フォーマットの文字列をパースし、Gura のオブジェクトを生成します。

yaml.read(stream:stream:r)

ストリームから YAML フォーマットの文字列を読み取り、Gura のオブジェクトを生成します。

yaml.write(stream:stream:w, obj):reduce

obj の内容を YAML フォーマットの文字列にしてストリームに出力します。

31.4. stream クラスの拡張

31.4.1. インスタンスメソッド

stream#yamlread()

ストリームから YAML フォーマットの文字列を読み取り、Gura のオブジェクトを生成します。

stream#yamlwrite(obj):reduce

obj の内容を YAML フォーマットの文字列にしてストリームに出力します。

32. uuid モジュール

32.1. 概要

UUID を生成します。使用するには import 関数を使って uuid モジュールをインポートします。

32.2. モジュール関数

uuid.generate():[upper]

UUID を生成し、文字列にして返します。アトリビュート:upper をつけると、16 進数の A から F までの文字を大文字にします。

33. mswin モジュール

33.1. 概要

Microsoft Windows で提供される機能を扱うモジュールです。使用するには import 関数を使って mswin モジュールをインポートします。

Windows COM インターフェースへのアクセスや、レジストリ操作が可能になります。

33.2. mswin.ole クラス

33.2.1. インスタンスの生成

```
mswin.ole(progid:string):map:[connect,no_const]
```

指定の ProgID に対応する COM サーバを生成し、その COM サーバへのインターフェースをインスタンスメソッドとして備えた mswin.ole インスタンスを返します。アトリビュート :connect を指定すると、すでに存在する COM サーバへの接続を行います。

デフォルトでは、TypeInfo 中に定数値があるとき、これらを mswin.ole インスタンス中にプロパティとしてとりこみますが、アトリビュート :no_const をつけるとこの処理を省きます。

33.3. mswin.regkey クラス

33.3.1. 概要

Windows のレジストリを扱うクラスです。

33.3.2. 定義済みインスタンス

モジュール mswin には、レジストリのルートキーを参照する regkey 型のインスタンスが以下のようにあらかじめ定義されています。

```
mswin.HKEY_CLASSES_ROOT
mswin.HKEY_CURRENT_CONFIG
mswin.HKEY_CURRENT_USER
mswin.HKEY_LOCAL_MACHINE
mswin.HKEY_USERS
mswin.HKEY_PERFORMANCE_DATA
mswin.HKEY_DYN_DATA
```

33.3.3. インスタンスメソッド

```
mswin.regkey#createkey(subkey:string,
                       option?:number, samDesired?:number):map {block?}
```

サブキーを作成します。subkey にキーの名前を指定します。

option には以下のいずれかの値を指定します。

- mswin.REG_OPTION_NON_VOLATILE

- mswin.REG_OPTION_VOLATILE
- mswin.REG_OPTION_BACKUP_RESTORE

samDesired には以下のセキュリティアクセスマスク値を組み合わせた値を指定します。

- mswin.KEY_CREATE_LINK
- mswin.KEY_CREATE_SUB_KEY
- mswin.KEY_ENUMERATE_SUB_KEYS
- mswin.KEY_EXECUTE
- mswin.KEY_NOTIFY
- mswin.KEY_QUERY_VALUE
- mswin.KEY_SET_VALUE
- mswin.KEY_ALL_ACCESS
- mswin.KEY_READ
- mswin.KEY_WRITE

mswin.regkey#deletekey(subkey:string):map:void

指定のキー subkey を削除します。

mswin.regkey#deletevalue(valueName:string):map:void

指定の値 valueName を削除します。

mswin.regkey#enumkey(samDesired?:number):[openkey] {block?}

デフォルトの動作では、サブキー名の一覧を得るイテレータを生成します。このとき、samDesired の値は意味を持ちません。

アトリビュート:openkey をつけると、サブキーをオープンし、そのキーに対応する mswin.regkey インスタンスを得るイテレータになります。このとき、samDesired はオープンするキーに対するセキュリティアクセスマスク値になります。

mswin.regkey#enumvalue()

レジストリエントリの名前の一覧を得るイテレータを生成します。

mswin.regkey#openkey(subkey:string, samDesired?:number):map {block?}

サブキーsubkey をオープンします。samDesired はオープンするキーに対するセキュリティアクセスマスク値です。

mswin.regkey#queryvalue(valueName?:string):map

レジストリエントリのデータを取得します。valueName にレジストリエントリの名前を指定して実行すると、データ内容が返ります。指定の名前のレジストリエントリが無い場合はエラーになります。

mswin.regkey#setvalue(valueName:string, data:nomap):map

レジストリエントリのデータを設定します。valueName はレジストリエントリの名前、data は設定するデータです。

33.4. COM について

33.4.1. COM サーバへの接続

COM は Microsoft が開発したアプリケーションインターフェースの仕様です。Microsoft Word や Excel、Internet Explorer が COM をサポートしており、これらのアプリケーションの動作をすべて外部からコントロールすることができます。このような COM を外部に提供しているアプリケーションや DLL を COM サーバと呼びます。

mswin.ole で mswin.ole インスタンスを生成すると、指定した COM サーバへの接続を確立し、COM サーバが提供するメソッドやプロパティを動的に作成します。

以下は Microsoft Excel を起動し、既存のファイルをオープンする例です。

```
import(mswin)
mswin.ole('Excel.Application') {|app|
    app.Visible = 1
    app.Workbooks.Open(path.absname('hoge.xls'))
}
```

以下は Microsoft Word を起動し、既存のファイルをオープンする例です。

```
import(mswin)
mswin.ole('Word.Application') {|app|
    app.Visible = 1
    app.Documents.Open(path.absname('hoge.doc'))
}
```

33.4.2. プロパティの取得

mswin.ole インスタンスでプロパティ名をメンバとして参照すると、OLE プロパティの取得を行います。このとき、値の型を以下のように変換します。（注: 2012/06 現在、リストには対応していません）

OLE 型	スクリプトの型	説明
VT_UI1	number	1 バイト符号なし整数
VT_I2	number	2 バイト符号付き整数
VT_I4	number	4 バイト符号付き整数
VT_R4	number	4 バイト浮動小数点数値
VT_R8	number	8 バイト浮動小数点数値
VT_BOOL	boolean	ブーリアン値。0 のとき false、それ以外を true にします。
VT_DATE	datetime	時刻。タイムゾーンとしてローカルタイムを設定します
VT_BSTR	string	文字列
VT_DISPATCH	mswin.ole	OLE ディスパッチャ
VT_DECIMAL		(未対応)
VT_ERROR		(未対応)
VT_CY		(未対応)
VT_UNKNOWN		(未対応)

VT_VARIANT		(未対応)

COM へのアクセスは、プロパティ名に対応する DispID を指定して、DISPATCH_PROPERTYGET を実行しています。

33.4.3. プロパティの設定

mswin.ole インスタンスでプロパティ名をメンバにしたものに対して代入をすると、OLE プロパティの設定を行います。このとき、値の型を以下のように変換します。

スクリプトの型	OLE 型	説明
number (整数値)	VT_I4	4 バイト符号付き整数
number (実数値)	VT_R8	8 バイト浮動小数点数値
string	VT_BSTR	文字列
boolean	VT_BOOL	ブーリアン値。true のとき –1、false のとき 0 を設定します
list	VT_ARRAY	リスト
mswin.ole	VT_DISPATCH	OLE ディスパッチャ
datetime	VT_DATE	時刻。タイムゾーンを無視し設定日時をそのまま反映させます

COM へのアクセスは、プロパティ名に対応する DispID を指定して、DISPATCH_PROPERTYPUT を実行しています。

33.4.4. メソッドの実行

mswin.ole インスタンスのメンバを引数リストつきで評価すると、引数リスト内の値を OLE タイプに変換してから OLE メソッドを実行します。実行した結果得られた値をスクリプトの型に変換し、評価値として返します。

COM へのアクセスは、メソッド名に対応する DispID を指定して (DISPATCH_METHOD | DISPATCH_PROPERTYGET) を実行しています。

33.4.5. イテレータの生成

イテレータを期待している文中に mswin.ole インスタンスを指定したとき、内包している OLE オブジェクトがイテレータに対応していれば、適切なイテレータを生成します。以下は、Excel ワークブック中の全てのワークシート名を表示する例です。

```
import(mswin)
mswin.ole('Excel.Application') {|app|
    app.Visible = 1
    wb = app.Workbooks.Open(path.absname('hoge.xlsx'))
    for (ws in wb.WorkSheets) {
        println(ws.Name)
    }
}
```

COM へのアクセスは、DispID に DISPID_NEWENUM を指定して DISPATCH_METHOD を実行しています。

34. midi モジュール

34.1. 概要

35. lets_module モジュール

35.1. 概要

バイナリモジュールの C++ソースコードとビルド用スクリプトのひな型を作成します。

```
$ gura -i lets_module hoge
```

36. modbuild モジュール

36.1. 概要

バイナリモジュールをビルドするためのモジュールです。使用するには import 関数を使って modbuile モジュールをインポートします。

以下は Module_hoge.cpp から hoge.gurd をビルドするスクリプトの例です。

```
import(modbuild)
builder = modbuild.Builder()
builder.build('hoge', ['Module_hoge.cpp'])
```

36.2. modbuild.Builder クラス

36.3. インスタンスプロパティ

プロパティ	データ型	R/W	説明
cflags	string	R/W	コンパイラオプション
incDirs	list	R/W	インクルードファイルのディレクトリ
ldflags	list	R/W	リンカオプション
precompile	string	R/W	Precompile するソースファイル名
progressFlag	Boolean	R/W	true のとき、コンパイル中のファイル名を表示します
hint	string	R/W	ビルドに失敗したときに表示するヒント文字列

36.4. インスタンスメソッド

modbuild.Builder#build(target:string, srcs[]:string)

バイナリモジュールをビルドします。

target にサフィックスを取り除いたモジュールファイル名を指定します。階層構造中のモジュールである場合、ディレクトリ名を含めたパス名を指定します。

srcs はコンパイルするソースファイルをリストで指定します。リスト中の最初のファイルをモジュールのメインファイルとして扱います。

37. gurcbuild モジュール

37.1. 概要

コンポジットファイルを作成するモジュールです。使用するには import 関数を使って gurdbuild モジュールをインポートします。

以下はコンポジットファイル hoge.gurc を作成するスクリプトの例です。

```
import(gurcbuild)
gurcbuild.build(['hoge.gura', 'image1.png', 'image2.png'])
```

37.2. モジュール関数

gurcbuild.build(pathNames[]:string, dirName?:string)

コンポジットファイルに格納するファイルを pathNames に指定します。pathNames の最初のファイルはスクリプトファイルでなくてはいけません。最初のファイル名のサフィックスを、.gurc にリネームしたものが出力するコンポジットファイルの名前になります。

コンポジットファイルはカレントディレクトリに生成されます。出力ディレクトリを変えたいときは引数 dirName を設定します。