Gura ライブラリリファレンス

Updated: January 15, 2014

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. 定義済み変数

変数	内容
*	iterator クラスのインスタンス 0.. に相当する値が定義されています。
-	nil クラスの nil 値が定義されています。
@rem	nil クラスの nil 値が定義されています。
__name__	現在実行しているスクリプトがメインのスクリプトファイルである場合、文字列 "__main__" が入ります。スクリプトがモジュール内である場合はそのモジュール名が入ります。
false	boolean クラスの偽値が定義されています。
nil	nil クラスの nil 値が定義されています。
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。
true	boolean クラスの真値が定義されています。

3. 組込み関数

3.1. 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。

3.2. 制御構文

3.2.1. 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は既存言語の制御構文のそれにあわせているので、動作内容が類推しやすくなっています。

3.2.2. 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。

ブロックパラメータの形式は `|idx:number|` で、idx に 0 から始まるループの回数が入ります。

スクリプト	実行結果
`repeat (10) {|i| printf(' %d', i)}`	`0 1 2 3 4 5 6 7 8 9`

`repeat { println('hello world') }`	`hello world`
	`hello world`
	`hello world`
	`:`

`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。

ブロックパラメータの形式は |idx:number, i0:number, i1:number, ..| で、idx に 0 から始まる全体のループの回数、i0, i1 .. に各イテレータの現在のインデクス値が入ります。

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]

3.2.3. 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。

3.2.4. 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')
}

3.2.5. 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 (error.ValueError) {|e|
    println('ValueError captured: ', e.text)
} catch (error.IOError) {|e|
    println('IOError captured: ', e.text)
} catch {|e|
    println('other error captured: ', e.text)
}
```

3.2.6. 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。

3.2.7. 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。

3.3. 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。

3.4. 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。

block をつけると、生成した参照を|cls:class| という形式のブロックパラメータで渡し、ブロックの内容を評価します。関数の戻り値はブロックの最後の評価値になります。

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。

block をつけると、オブジェクトへの参照を|obj| という形式のブロックパラメータで渡し、ブロックの内容を評価します。関数の戻り値はブロックの最後の評価値になります。

3.5. 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。

3.6. 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。

block をつけると、生成したリストを|list| という形式のブロックパラメータで渡し、ブロックの内容を評価します。関数の戻り値はブロックの最後の評価値になります。

3.7. ストリーム処理

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。

3.8. 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。

3.9. データ型処理

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。

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。

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)

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。

undef(`symbol+):[raise]

シンボルを未定義に設定します。アトリビュート :raise を指定すると、シンボルがすでに未定義のときはエラーになります。

3.10. 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。

3.11. 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。

3.12. 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。

4. プリミティブクラス

4.1. nil クラス

4.1.1. 概要

無効値を表すクラスです。

4.1.2. インスタンスの生成

グローバル変数として、nil という名前のインスタンスが定義されています。

4.1.3. スクリプト内での扱い

スクリプト内において、nil は無効値として扱われます。

4.2. boolean クラス

4.2.1. 概要

ブール型を扱うクラスです。

4.2.2. インスタンスの生成

グローバル変数として true と false という名前のインスタンスが定義されており、それぞれが真と偽を表します。

4.2.3. 真偽の扱い

false と nil が偽、その他の値はすべて真として扱われます。

4.3. complex クラス

4.3.1. 概要

複素数を扱うクラスです。

4.3.2. インスタンスの生成

リテラル j をコード中に記述すると、complex インスタンスが生成されます。また、コンストラクタ関数 complex を使って生成することもできます。

```
complex(real:number, imag?:number):map {block?}
```

引数 real および imag にそれぞれ実数部と虚数部を指定し、complex インスタンスを生成します。引数 imag を省略すると、虚数部が 0 になります。

block をつけると、生成したインスタンスへの参照を|comp:complex| という形式のブロックパラメータで渡し、ブロックの内容を評価します。関数の戻り値はブロックの最後の評価値になります。

```
complex.polar(abs:number, arg:number):map:[deg] {block?}
```

引数 abs に絶対値、arg 偏角をラジアン値で指定し、complex インスタンスを生成します。アトリビュート :deg をつけると、arg を degree 値で指定することができます。

block をつけると、生成したインスタンスへの参照を|comp:complex| という形式のブロックパラメータで渡し、ブロックの内容を評価します。関数の戻り値はブロックの最後の評価値になります。

4.3.3. インスタンスプロパティ

プロパティ	型	R/W	説明
abs	number	R	複素数の絶対値を返します。
arg	number	R	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。
imag	number	R	複素数の虚数成分を返します。
norm	number	R	複素数のノルム値を返します。
real	number	R	複素数の実数成分を返します。

4.4. rational クラス

4.4.1. 概要

分数を扱うクラスです。

4.4.2. インスタンスの生成

数値リテラルにサフィックス r をつけると、その数値を分子にした rational インスタンスを生成します。たとえば、3r は rational(3, 1) を実行したものと同じインスタンスを生成します。

また、以下のコンストラクタ関数 rational を使ってインスタンスを生成することができます。

```
rational(numer:number, denom?:number):map {block?}
```

引数 numer および denom にそれぞれ分子と分母を指定して rational インスタンスを生成します。引数 denom を省略すると分母が 1 になります。

block をつけると、生成したインスタンスへの参照を|ratio:rational| という形式のブロックパラメータで渡し、ブロックの内容を評価します。関数の戻り値はブロックの最後の評価値になります。

4.4.3. インスタンスプロパティ

プロパティ	型	R/W	説明
denom	number	R	分母を返します。
numer	number	R	分子を返します。

4.4.4. インスタンスメソッド

```
rational#reduce()
```

分子と分母を通分した結果を返します。

4.5. number クラス

4.5.1. 概要

数を扱うクラスです。内部は 64 ビット浮動小数点数値で表現されます。

4.5.2. インスタンスプロパティ

プロパティ	型	R/W	説明
abs	number	R	絶対値を返します。
arg	number	R	常に 0 を返します。
imag	number	R	常に 0 を返します。
norm	number	R	二乗した数値を返します。
real	number	R	自分自身を返します。

4.6. string クラス

4.6.1. 概要

文字列を扱うクラスです。

4.6.2. 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。

4.6.3. 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#binary()`

文字列の内容を binary 型に変換した結果を返します。

`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。

block をつけると、文字コード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 |ch:number, idx:number| で、ch に文字コード、idx に 0 から始まるインデクス番号が入ります。

`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。

block をつけると、一行の文字列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 に一行ごとの文字列、idx に 0 から始まるインデクス番号が入ります。

`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():[quote]`

HTML 書式で処理ができるよう、"&"、"<" および ">" をそれぞれ "&"、"<" および">" に変換します。アトリビュート :quote をつけると、さらにダブルクオーテーション """ を """ に変換します。

`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#fold(len:number, step?:number):[neat] {block?}`

文字列を len 文字ずつ分けた結果を要素に持つイテレータを返します。step を指定すると、step 文字

間隔で分けます。アトリビュート :neat をつけると、最後に切り分けた文字列が len 文字に満たない場合はそれを結果に含めません。

block をつけると、分割した文字列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 |str:string, idx:number| で、str に分割した文字列、idx に 0 から始まるインデクス番号が入ります。

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。stream を省略した場合は標準出力に出力します。

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。stream を省略した場合は標準出力に出力します。

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。

block をつけると、生成したストリームへの参照を|stream:stream| という形式のブロックパラメータで渡し、ブロックの内容を評価します。関数の戻り値はブロックの最後の評価値になります。

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。

block をつけると、切り分けた文字列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 |str:string, idx:number| で、str に切り分けた文字列、idx に 0 から始まるインデクス番号が入ります。

```
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#template():[lasteol,noindent] {block?}
```

文字列中に記述されたスクリプトを評価し、template インスタンスを生成します。

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。

block をつけると、生成した template インスタンスへの参照を|tmpl:template| という形式のブロックパラメータで渡し、ブロックの内容を評価します。関数の戻り値はブロックの最後の評価値になります。

```
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。

4.7. symbol クラス

4.7.1. 概要

シンボル値を扱うクラスです。

4.7.2. インスタンスの生成

単一のシンボルトークンにバッククオート ` をつけると symbol インスタンスになります。

5. 組込みクラス

5.1. object クラス

5.1.1. 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。

5.1.2. インスタンスの生成

```
object() {block?}
```

object 型インスタンスを生成します。
block をつけると、生成した object インスタンスへの参照を|obj| という形式のブロックパラメータで渡し、ブロックの内容を評価します。関数の戻り値はブロックの最後の評価値になります。

5.2. args クラス

5.2.1. 概要

関数に渡された引数情報を扱うクラスです。

5.2.2. 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。

5.2.3. 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

5.2.4. 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。

5.3. audio クラス

5.3.1. 概要

オーディオデータを扱うクラスです。

5.4. binary クラス

5.4.1. 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たりすることができます。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。

5.4.2. 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*) {block?}

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。

5.4.3. クラスメソッド

binary.pack(format:string, values*):map {block?}

引数 fomat で指定したフォーマットに基づいて、values 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、引数列 values 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。

5.4.4. 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) {block?}`

バイナリデータにアクセスする pointer インスタンスを返します。

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, values*:number):[nil]

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。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が、アトリビュート :nil をつけると範囲外では nil を返すようになります。

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。

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、引数列 values 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。

```
binary#unpacks(format:string, values*: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 の内容は binary#unpack と同じです。

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。

5.5. codec クラス

5.5.1. 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。

5.5.2. 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。

5.5.3. 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

5.5.4. 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。

5.5.5. 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。

5.6. `color` クラス

5.6.1. 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。

5.6.2. インスタンスの生成

`color(name, alpha?:number):map {block?}`

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 {block?}`

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。

5.6.3. 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

5.6.4. 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`、アルファ値 `255` を持った `color` インスタンスです

Maroon	color	R	色要素 #800000、アルファ値 255 を持った color インスタンスです
Green	color	R	色要素 #008000、アルファ値 255 を持った color インスタンスです
Olive	color	R	色要素 #808000、アルファ値 255 を持った color インスタンスです
Navy	color	R	色要素 #000080、アルファ値 255 を持った color インスタンスです
Purple	color	R	色要素 #800080、アルファ値 255 を持った color インスタンスです
Teal	color	R	色要素 #008080、アルファ値 255 を持った color インスタンスです
Gray	color	R	色要素 #808080、アルファ値 255 を持った color インスタンスです
Silver	color	R	色要素 #c0c0c0、アルファ値 255 を持った color インスタンスです
Red	color	R	色要素 #ff0000、アルファ値 255 を持った color インスタンスです
Lime	color	R	色要素 #00ff00、アルファ値 255 を持った color インスタンスです
Yellow	color	R	色要素 #ffff00、アルファ値 255 を持った color インスタンスです
Blue	color	R	色要素 #0000ff、アルファ値 255 を持った color インスタンスです
Fuchsia	color	R	色要素 #ff00ff、アルファ値 255 を持った color インスタンスです
Aqua	color	R	色要素 #00ffff、アルファ値 255 を持った color インスタンスです
White	color	R	色要素 #ffffff、アルファ値 255 を持った color インスタンスです
Zero	color	R	色要素 #000000、アルファ値 0 を持った color インスタンスです

5.6.5. 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

5.6.6. 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。

5.6.7. 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

5.7. datetime クラス

5.7.1. 概要

時刻を表すクラスです。

5.7.2. インスタンスの生成

```
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 {block?}
```

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。

5.7.3. クラスプロパティ

プロパティ	型	R/W	内容
Sunday	number	R/W	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	R/W	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	R/W	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	R/W	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	R/W	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	R/W	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	R/W	土曜日を表すインデクス番号 6 が代入されています

5.7.4. 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

5.7.5. クラスメソッド

```
datetime.now():[utc] {block?}
```

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。

```
datetime.time(hour:number => 0, minute:number => 0,
              sec:number => 0, usec:number => 0):map {block?}
```

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。

```
datetime.today():[utc] {block?}
```

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。

```
datetime.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。

```
datetime.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。

```
datetime.parse(str:string):map {block?}
```

日付文字列を解析し、その結果を datetime インスタンスとして返します。対応している日付文字列のフォーマットは以下の通りです。

- W3C の仕様で使われるフォーマット
- RFC で定義される HTTP の仕様で使われるフォーマット
- C 言語の asctime 関数のフォーマット

```
datetime.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。

5.7.6. 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。

5.8. declaration クラス

5.8.1. 概要

function インスタンスの引数宣言を表すクラスです。

5.8.2. 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。

5.8.3. 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

5.9. dict クラス

5.9.1. 概要

辞書型のデータを扱うクラスです。

5.9.2. 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, ..}
```

5.9.3. 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。

5.10. directory クラス

5.10.1. 概要

列挙が可能なパス内を走査するためのクラスです。通常 path.dir() や path.walk() などの関数とともに用いられます。

5.10.2. インスタンスの生成

directory(pathname:string):map {block?}

directory 型インスタンスを生成します。

5.11. environment クラス

5.11.1. 概要

変数などを格納している environment の内容を操作するクラスです。

5.11.2. 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。

5.11.3. インスタンスメソッド

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。

5.12. error クラス

5.12.1. 概要

エラー内容を扱うクラスです。

5.12.2. インスタンスの生成

- 関数 catch のブロックパラメータとして渡されます。

5.12.3. クラスプロパティ

プロパティ	型	R/W	説明
ArgumentError	error	R	error インスタンス
ArithmeticError	error	R	error インスタンス
AttributeError	error	R	error インスタンス
CodecError	error	R	error インスタンス
CommandError	error	R	error インスタンス
DeclarationError	error	R	error インスタンス
FormatError	error	R	error インスタンス
IOError	error	R	error インスタンス
ImportError	error	R	error インスタンス
IndexError	error	R	error インスタンス
IteratorError	error	R	error インスタンス
KeyError	error	R	error インスタンス
MemberAccessError	error	R	error インスタンス
MemoryError	error	R	error インスタンス
NameError	error	R	error インスタンス
NotImplementedError	error	R	error インスタンス
OutOfRange	error	R	error インスタンス
ResourceError	error	R	error インスタンス
RuntimeError	error	R	error インスタンス
SyntaxError	error	R	error インスタンス
SystemError	error	R	error インスタンス
TypeError	error	R	error インスタンス
ValueError	error	R	error インスタンス
ZeroDivisionError	error	R	error インスタンス

5.12.4. インスタンスプロパティ

プロパティ	型	R/W	説明
lineno	number	R	ソース中、エラーを発生した式が記述されている行番号を返します。
linenobtm	number	R	ソース中、エラーを発生した式が記述されている最後の行番号を返します。
postext	string	R	エラーを発生した式が記述されている位置を文字列で返します。
source	string	R	エラーを発生したコードの情報源がストリームの場合はそのパス名を返します。それ以外の場合以下の文字列を返します。 <interactive>　対話形式で入力された式

			<cmdline>　コマンドラインで指定された式 <ole>　OLE 内で記述された式
text	string	R	エラー文字列を返します。
trace	iterator	R	エラーを発生した式からルートまでのトレース情報です。expr を要素に持つイテレータを返します。

5.13. expr クラス

5.13.1. 概要

Gura スクリプトの構文木を扱うクラスです。

5.13.2. 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。
以下のコンストラクタを使うと、ストリームから expr インスタンスを生成できます。

```
expr(src:stream:r):map {block?}
```

ストリームからスクリプト文字列を読み込んで解析し、expr インスタンスを返します。

5.13.3. 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、およびシンボルの一覧を以下に示します。

Expr 要素	判定メソッド	シンボル
Assign	expr#isassign()	assign
Binary	expr#isbinary()	(抽象要素)
BinaryOp	expr#isbinaryop()	binaryop
Block	expr#isblock()	block
Caller	expr#iscaller()	caller
Compound	expr#iscompound()	(抽象要素)
Container	expr#iscontainer()	(抽象要素)
Member	expr#ismember()	member
Indexer	expr#isindexer()	indexer
Lister	expr#islister()	lister
Quote	expr#isquote()	quote
String	expr#isstring()	string
Symbol	expr#issymbol()	symbol
Unary	expr#isunary()	(抽象要素)
UnaryOp	expr#isunaryop()	unaryop
Value	expr#isvalue()	value

5.13.4. インスタンスプロパティ

プロパティ	型	R/W	説明
block	expr	R	Caller 要素が持つブロックの内容を返します。 ブロックが存在しない場合 nil を返します。
blockparam	iterator	R	Caller 要素が持つブロックパラメータの内容を expr 型で返すイテレータを生成します。 ブロックパラメータが存在しない場合 nil を返します。
car	expr	R	Compound 要素が持つ car の内容を返します。
cdr	iterator	R	Compound 要素が持つ cdr の内容を expr 型で返すイテレータを生成します。
child	expr	R	Unary 要素が持つ child の内容を返します。
children	iterator	R	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。
left	expr	R	Binary 要素の左側要素の内容を返します。
lineno	number	R	ソース中、式が記述されている行番号を返します。
linenobtm	number	R	ソース中、式が記述されている最後の行番号を返します。
operator	operator	R	UnaryOp、BInaryOp および Assign が持つオペレータインスタンスを返します。 UnaryOp と BinaryOp に対しては常に有効な値を返しますが、Assign については、オペレータが存在しない場合 nil を返します。
postext	string	R	式が記述されている位置を文字列で返します。
right	expr	R	Binary 要素の右側要素の内容を返します。
source	string	R	ソースがストリームの場合はそのパス名を返します。それ以外の場合以下の文字列を返します。 <interactive> 対話形式で入力された式 <cmdline> コマンドラインで指定された式 <ole> OLE 内で記述された式
string	string	R	String 要素の文字列データを返します。
symbol	symbol	R	Symbol 要素のシンボル値を返します。
trailer	expr	R	Caller 要素が持つトレーラーの内容を返します。
typename	string	R	タイプ名を返します。これは typesym の内容を文字列にしたものです。
typesym	symbol	R	タイプシンボルを返します。
value	any	R	Value 要素の値を返します。

5.13.5. クラスメソッド

```
expr.parse(script:string) {block?}
```

文字列の内容をスクリプトとみなして解析し、expr インスタンスを返します。

5.13.6. インスタンスメソッド

`expr#eval(env?:environment)`

expr の内容を現在の環境で評価します。引数 env を指定すると、その環境で expr を評価します。

`expr#textize(style?:symbol)`

expr の内容をスクリプトに変換したものを文字列にして返します。

style は以下のうちのひとつを指定します。style を省略した場合は`fancy が選択されます。

`brief 以外で出力したスクリプトは通常の Gura スクリプトとして扱うことができます。

style	説明
`crammed	空白などをつめた形式
`oneline	改行を含まない、一行で表現した形式
`brief	ブロックの内容を省略した形式
`fancy	改行やインデントを用いて可読性を高めた形式

`expr#tofunction(`args*)`

指定した引数列を持つ関数に変換します。

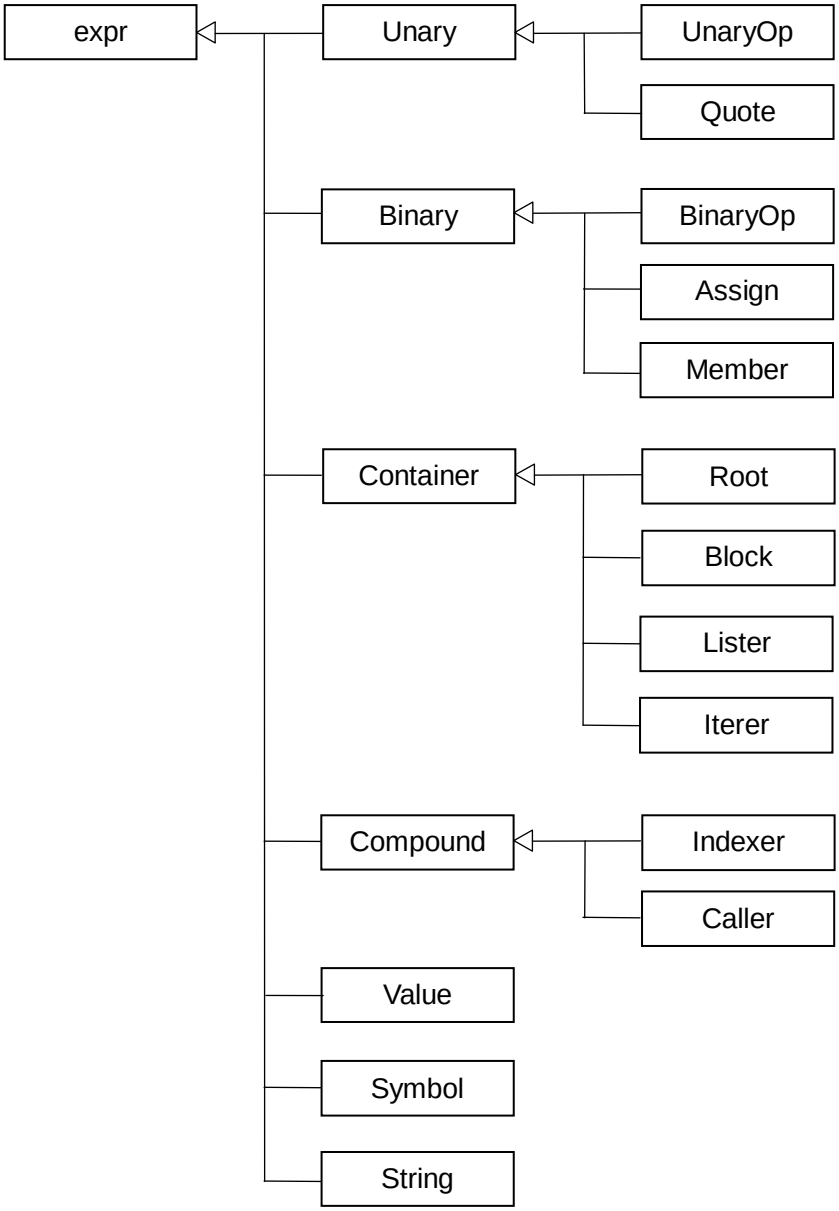
`expr#unquote()`

expr の内容が quote されているとき、それを取り除きます。

`expr#write(dst:stream:w, style?:symbol)`

引数 dst で指定したストリームに expr の内容をスクリプトに変換したものを出力します。style には expr#textize と同じシンボルを指定します。

5.13.7. 式を構成する要素



5.14. `function` クラス

5.14.1. 概要

関数を扱うクラスです。

5.14.2. 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。

5.14.3. インスタンスプロパティ

プロパティ	型	R/W	説明
`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/W	関数がスクリプトで定義されている場合、その本体を返します。関数がC++で記述されている場合は `nil` を返します。
`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。
`help`	`string`	R/W	関数に登録された `help` インスタンスを返します。
`symbol`	`symbol`	R/W	関数のシンボルを返します
`name`	`string`	R	関数の名前を文字列で返します

5.14.4. インスタンスメソッド

`function#addhelp(lang:symbol, format:string, help:string):map`

関数にヘルプ情報を追加します。

引数 `lang` はヘルプの記述言語を表すシンボルで、たとえば英語の場合は`` `en ``、日本語の場合は`` `ja `` を指定します。引数 `format` はヘルプ文字列のフォーマットを指定します。現在 `'markdown'` のみが指定可能です。引数 `help` にヘルプ文字列を渡します。

ヘルプ情報は異なる `lang` ごとに格納されます。以前と同じ `lang` に対してこのメソッドを実行すると、ヘルプ情報を上書きします。複数のヘルプ情報がある場合、最初に追加したヘルプ情報がデフォルトとして使用されます。

`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

`function#gethelp(lang?:symbol):map`

関数のヘルプ情報を help インスタンスで返します。引数 lang にはテキストの記述言語のシンボルを指定します。省略した場合はデフォルトの記述言語が指定されます。

`function#help(lang?:symbol):map:void`

関数のヘルプをコンソール画面に表示します。引数 lang にはテキストの記述言語のシンボルを指定します。省略した場合はデフォルトの記述言語が指定されます。

5.15. help クラス

5.15.1. 概要

ヘルプを扱うクラスです。

5.15.2. インスタンスプロパティ

プロパティ	型	R/W	説明
format	string	R	テキストの記述フォーマット。現在 'markdown' のみが有効です。
lang	symbol	R	テキストの言語をシンボルで指定返します。代表的なものを以下に示します。 ● `en 英語 ● `ja 日本語
text	string	R	テキスト本体

5.16. image クラス

5.16.1. 概要

イメージデータを扱うクラスです。

5.16.2. 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。

5.16.3. インスタンスプロパティ

プロパティ	型	R/W	説明
format	symbol	R	イメージの内部フォーマットを`rgb または `rgba で返します。
height	number	R	イメージの高さをピクセル単位で返します。
palette	palette	R/W	イメージに登録されているpaletteインスタンスを返します。パレットがない場合はnilを返します。
width	number	R	イメージの幅をピクセル単位で返します。

5.16.4. インスタンスメソッド

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

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#blur(radius:number, sigma:number) {block?}
```

```
image#clear():reduce
```

イメージ全体をゼロ値で埋めます。これはimage#fillにcolor.Zeroを渡した結果と同じです。

```
image#crop(x:number, y:number, width?:number, height?:number):map {block?}
```

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

t.b.d.

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

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:reduce
```

指定の範囲を指定した色で塗りつぶします。

```
image#flip(orient:symbol):map {block?}
```

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#grayscale() {block?}
イメージをグレースケールにした image インスタンスを生成して返します。

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:reduce
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) {block?}
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):reduce
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] {block?}
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 {block?}

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#scan(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。

シンボル	説明
`left_top_horz	左上から水平方向に走査します
`left_top_vert	左上から垂直方向に走査します
`left_bottom_horz	左下から水平方向に走査します
`left_bottom_vert	左下から垂直方向に走査します
`right_top_horz	右上から水平方向に走査します
`right_top_vert	右上から垂直方向に走査します
`right_bottom_horz	右下から水平方向に走査します
`right_bottom_vert	右下から垂直方向に走査します

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#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):reduce

t.b.d.

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

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。

5.17. iterator クラス

5.17.1. 概要

イテレータを扱うクラスです。

5.17.2. 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。

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。

`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。

`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?}`

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。

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。

5.17.3. 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。

5.18. list クラス

5.18.1. 概要

リストを扱うクラスです。

5.18.2. 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。

5.18.3. 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 ${}_{m}C_{n}$ 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#cycle(n?:number) {block?} / iterator#cycle(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#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#contains(value) / iterator#contains(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#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。

5.19. `matrix` クラス

5.19.1. 概要

行列を扱うクラスです。

5.19.2. 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 `0` 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, …}}`	

5.19.3. 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。

5.19.4. 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。

5.19.5. 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#roundoff(threshold:number => 1e-10)`

指定した値ですべての要素を丸めます。

`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。

5.20. operator クラス

5.20.1. 概要

演算子を扱うクラスです。

5.20.2. インスタンスの生成

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

オペレータシンボルに対応する operator インスタンスを返します。

5.20.3. 関数形式による評価

operator インスタンスに引数リストをつけて実行すると、その評価値を返します。このとき、引数の数が一つの場合は単項演算子として、二つの場合は二項演算子として評価します。

以下の例は 3 + 4 を実行したときと同じ結果を返します。

スクリプト

```
op = operator(`+)
op(3, 4)
```

5.20.4. インスタンスメソッド

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

オペレータに処理を登録します。

左辺の型を type_l、右辺の型を type_r に指定すると、二項演算子の処理として block の内容を登録します。ブロックパラメータの形式は |value_l, value_r| で、value_l に左辺の値、value_r に右辺の値が渡されます。

変数 type_l のみ指定した場合は単項演算子の処理として block の内容を登録します。このときのブロックパラメータの形式は |value| です。

以下の例は、二項演算子 "-" の左辺および右辺に string 型の値が指定されたときの処理を登録します。

スクリプト

```
operator(`-).assign(`string, `string) {|str, strSub|
    str.replace(strSub, '')
}
```

operator#entries(type?:symbol)

オペレータで処理できる型の一覧を取得します。

引数を省略するか、引数 type に`binary を指定すると、operator インスタンスに二項演算子としてアサインされている型シンボルのペアのリストを返します。引数 type に`unary を指定すると、単項演算子と

してアサインされている型シンボルのリストを返します。

5.21. palette クラス

5.21.1. 概要

色のパレットを扱うクラスです。

5.21.2. 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。

5.21.3. 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。

5.22. pointer クラス

5.22.1. 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。

5.22.2. インスタンスの生成

binary#pointer メソッドで生成します。

5.22.3. 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, values+):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, values*:number):[nil,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が、アトリビュート :nil をつけると範囲外では nil を返すようになります。

format 中に記述する指定子については 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, values*:number)

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 の内容は pointer#unpack と同じです。

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。

5.23. semaphore クラス

5.23.1. 概要

セマフォを扱うクラスです。

5.23.2. インスタンスの生成

semaphore()

semaphore 型インスタンスを生成します。

5.23.3. インスタンスメソッド

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。

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。

5.24. stream クラス

5.24.1. 概要

ストリームデータを扱うクラスです。

5.24.2. インスタンスの生成

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

引数pathname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。

5.24.3. インスタンスメソッド

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#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.copy(src:stream:r, dst:stream:w, bytesunit:number => 65536)
:map:void:[finalize] {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。
アトリビュート :finalize を指定すると、ストリーム dst をクローズしてストリーム src の属性をストリーム dst にコピーします。

```
stream#copyfrom(src:stream:r, bytesunit:number => 65536)
                                        :map:reduce:[finzlie] {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。
アトリビュート :finalize を指定すると、ストリームインスタンスをクローズしてストリーム src の属性をコピーします。

```
stream#copyto(dst:stream:w, bytesunit:number => 65536)
                                        :map:reduce:[finalize] {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。
アトリビュート :finalize を指定すると、ストリーム dst をクローズしてストリームインスタンスの属性をコピーします。

```
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#deserialize()
```

ストリームを読み込み、デシリアライズした結果を返します。ストリームの内容は stream#serialize の変換結果である必要があります。

```
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#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#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#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。

戻り値として `stream` インスタンスを返します。

`stream#serialize(value):void`

引数 `value` の内容をシリアライズしてストリームに書き込みます。

`stream#setcodec(codec:codec:nil):reduce`

ストリームでテキストデータの読み書きの際に扱う文字コードの `codec` インスタンスを指定します。`nil` を指定すると文字コードの変換は行いません。

戻り値として stream インスタンスを返します。

stream#tell()

ストリームのシーク位置を返します。

stream#write(buff:binary, len?:number):reduce

ストリームにバイナリデータを書き込みます。引数 len を指定すると、そのバイト数だけ書き込みを行います。

5.24.4. 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。

5.25. template クラス

5.25.1. 概要

テンプレートを扱うクラスです。テンプレートの詳細については「Gura 言語マニュアル」を参照ください。

5.25.2. インスタンスの生成

template インスタンスを生成するには string#template メソッドを使うか、以下のコンストラクタを実行します。

template(src?:stream:r):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、template インスタンスを生成します。引数 src を省略すると、空の template インスタンスを生成します。

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。

5.25.3. インスタンスメソッド

`template#parse(str:string):void:[lasteol,noindent]`

テンプレート文字列を読み込み、template インスタンスに内容を追加します。

`template#read(src:stream:r):void:[lasteol,noindent]`

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、template インスタンスに内容を追加します。

`template#render(dst?:stream:w)`

テンプレートの評価結果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。

5.26. timedelta クラス

5.26.1. 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

5.26.2. インスタンスの生成

`timedelta(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。

5.26.3. 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。

5.27. uri クラス

5.27.1. 概要

URI 文字列を扱うクラスです。

5.27.2. インスタンスの生成

`uri(str?:string):map {block?}`

文字列を URI として解釈した結果を持つ uri インスタンスを返します。

6. argopt モジュール

6.1. 概要

引数オプションを扱うモジュールです。使用するには import 関数を使って argopt モジュールをインポートします。

6.2. サンプル

スクリプト

```
import(argopt)

p = argopt.Parser()
p.addFlag('debug',     'd', 'debug mode')
p.addFlag('verbose',   'v', 'print messages verbosely')
p.addParam('level',    'l', 'specify a level', 'NUM', '0')
try {
    [cfg, argv] = p.parse(sys.argv)
} catch {|e|
    println(e.text)
    R'''
    usage: grep.gura [options] ...
    options:
      ${p.formatHelp():linefeed}
    '''.template().render(sys.stderr)
    sys.exit(1)
}
println('debug   = ', cfg['debug'])
println('verbose = ', cfg['verbose'])
println('level   = ', int(cfg['level']))
```

6.3. argopt.Parser クラス

6.3.1. インスタンスの生成

argopt.Parser() {block?}

argopt.Parser インスタンスを生成します。

6.3.2. インスタンスメソッド

argopt.Parser#parse(argv[]:string)

addFlag および addParam メソッドで登録されたオプション情報に基づいて引数列を解析し、結果を [cfg, argv] というリストで返します。cfg は longName をキーとした辞書、argv はオプション以外の引数列です。

addFlag で指定したオプションが引数列に存在すると、cfg 中の対応する要素に true を代入し、存在しない場合は nil を代入します。

addParam で指定したオプションが引数列に存在すると、cfg 中の対応する要素にその値を string 型で代入し、存在しない場合は addParam メソッドの defValue で指定した値を代入します。

argopt.Parser#addFlag(longName:string, shortName?:string, help?:string)

値をとらないオプションの情報を追加します。longName はロング形式のオプション名で、parse メソッドで返される辞書のキーとしても使われます。shortName は一文字からなるショート形式のオプション名です。help には formatHelp メソッドで返すヘルプ文字列を指定します。

```
argopt.Parser#addParam(longName:string, shortName?:string, help?:string,
                                    helpValue?:string, defValue?:string)
```

値をとるオプションの情報を追加します。longName はロング形式のオプション名で、parse メソッドで返される辞書のキーとしても使われます。shortName は一文字からなるショート形式のオプション名です。help と helpValue には formatHelp メソッドで返すヘルプ文字列を指定します。defValue はこのオプションが指定されなかったときに代入する値を指定します。defValue を指定しない場合は nil が使われます。

```
argopt.Parser#formatHelp(longNameFlag:boolean => true,
                                    shortNameFlag:boolean => true):[linefeed]
```

ヘルプ文字列を一行ずつ返すイテレータを生成します。longNameFlag を true にするとロング形式のオプション情報を文字列に加えます。shortNameFlag を true にするとショート形式のオプション情報を文字列に加えます。

アトリビュート:linefeed を付けると、各行の最後に改行コードを追加します。

7. bmp モジュール

7.1. 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。

7.2. サンプル

スクリプト

```
import(bmp)
img = image('hoge.bmp')
```

7.3. 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）

7.4. image クラスの拡張

7.4.1. 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。

8. bzip2 モジュール

8.1. 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/

8.2. サンプル

スクリプト

```
import(bzip2)
bzip2.writer('test.dat.bz2').copyfrom('test.dat')
bzip2.reader('test.dat.bz2').copyto('test2.dat')
```

8.3. 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, blockSize100k?:number) {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。

8.4. stream クラスの拡張

8.4.1. 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(blockSize100k?:number) {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。

9. cairo モジュール

「Gura モジュールリファレンス – cairo」を参照ください。

10. conio モジュール

10.1. 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。

10.2. サンプル

スクリプト

```
conio.clear()
[w, h] = conio.getwinsize()
conio.moveto(0, 0) {
    print('*' * w)
}
conio.moveto(0, 1 .. (h - 2)) {
    print('*', ' ' * (w - 2), '*')
}
conio.moveto(0, h - 1) {
    print('*' * w)
}
conio.waitkey():raise
```

10.3. 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

11. csv モジュール

11.1. 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。

11.2. サンプル

スクリプト

```
import(csv)
Record = struct(name:string, age:number, email:string)
records = Record * csv.read('records.csv')
printf('name:%s, age:%d, email:%s¥n',
          records:*name, records:*age, records:*email)
```

11.3. 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.read(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。

11.4. csv.writer クラス

11.4.1. 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。

11.4.2. 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。

11.5. stream クラスの拡張

11.5.1. インスタンスメソッド

`stream#csvread() {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。

12. curl モジュール

12.1. 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

12.2. サンプル

スクリプト

12.3. 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。

12.4. モジュール変数

12.5. モジュール関数

curl.version()

cURL のバージョン文字列を返します。

12.6. curl.easy_handle クラス

12.6.1. 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。

12.6.2. インスタンスメソッド

curl.easy_handle#escape(string:string):void

curl_easy_escape()

curl.easy_handle#getinfo(info:number)

curl_easy_getinfo()

```
curl.easy_handle#pause(bitmask:number):void
    curl_easy_pause()

curl.easy_handle#perform(stream?:stream:w):void
    curl_easy_perform()

curl.easy_handle#recv(buflen:number)
    curl_easy_recv()

curl.easy_handle#reset():void
    curl_easy_reset()

curl.easy_handle#send(buffer:binary)
    curl_easy_send()

curl.easy_handle#setopt(option:number, arg):void
    curl_easy_setopt()

curl.easy_handle#unescape(string:string):void
    curl_easy_unescape()
```

13. freetype モジュール

13.1. 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/

13.2. サンプル

スクリプト

```
import(freetype, ft)
img = image(`rgba, 400, 400, `white)
ft.font('arial.ttf') {|font|
    font.height = 64
    str = 'Hello World'
    [w, h] = font.calcsize(str)
    img.drawtext(font, (img.width - w) / 2, (img.height + h) / 2, str)
}
```

13.3. 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。

13.4. freetype.font クラス

13.4.1. 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。

13.4.2. 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。

13.4.3. 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。

13.4.4. 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。

13.5. freetype.Face クラス

13.5.1. 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。

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
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			

13.5.3. 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。

13.6. freetype.GlyphSlot クラス

13.6.1. 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		

13.6.2. 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。

13.7. 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。

13.8. freetype.Glyph クラス

13.9. freetype.Matrix クラス

13.9.1. 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。

13.9.2. 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。

13.10. `freetype.Vector` クラス

13.10.1. 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。

13.11. `image` クラスの拡張

13.11.1. 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。

14. fs モジュール

14.1. 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。

14.2. サンプル

スクリプト

14.3. 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。

14.4. パスのサーチ

path.dir、path.walk および 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。

14.5. 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.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

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

a	全てのユーザ

`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.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.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。

14.6. `fs.stat` クラス

14.6.1. 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

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. サンプル

スクリプト

```
import(gif)
g = gif.content()
g.addimage(['cell1.png', 'cell2.png', 'cell3.png'], 10)
g.write('anim.gif')
```

15.3. 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）

15.4. gif.content クラス

15.4.1. 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。

15.4.2. 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。

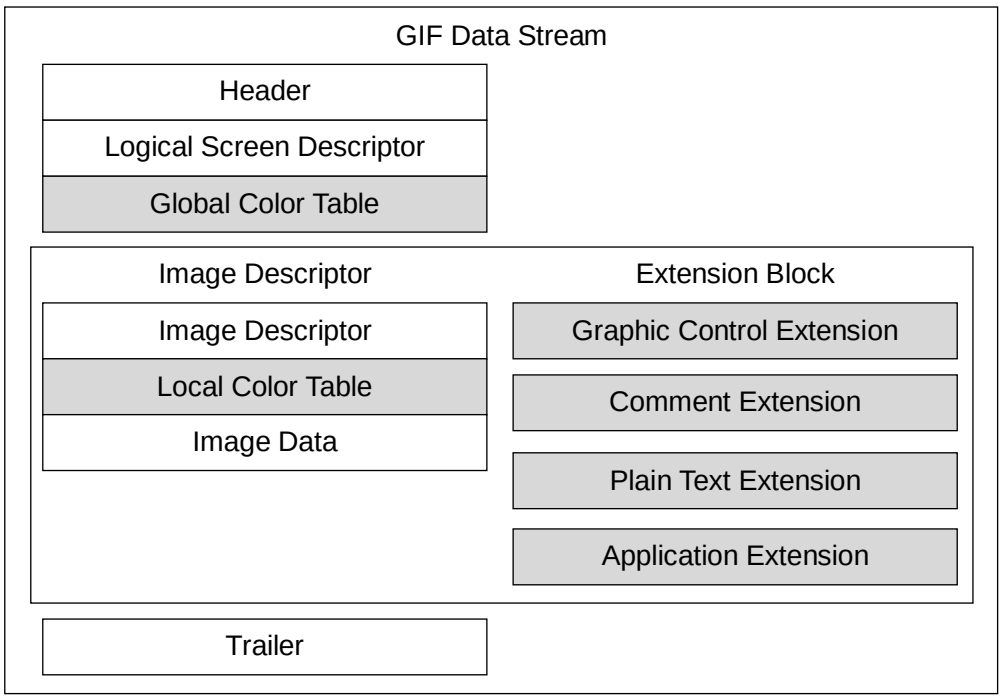


Image Descriptor と Extension Block は任意の数だけ GIF Data Stream に表れます。ただし、Graphic Control Extension は各 Image Descriptor の前に一つだけ置くことができます。

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。

15.4.3. 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。

15.4.4. 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。

15.4.5. 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。

15.4.6. 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。

15.4.7. 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

15.5. image クラスの拡張

15.5.1. 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。

15.5.2. 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

15.5.3. 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。

15.5.4. 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. glu モジュール

「Gura モジュールリファレンス – opengl」を参照ください。

17. gmp モジュール

17.1. 概要

多倍長数値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 gmp モジュールをインポートします。

Linux 環境においては、以下の URL で公開されている GMP ライブラリを内部で使用しています。

https://gmplib.org/

Windows 環境においては、以下の URL で公開されている MPIR ライブラリを内部で使用しています。

http://www.mpir.org/

17.2. サンプル

スクリプト
`import(gmp)`

17.3. モジュール関数

`gmp.gcd(num1:gmp.mpz, num2:gmp.mpz)`

num1 と num2 の最大公約数を gmp.mpz 型で返します。

17.4. gmp.mpz クラス

17.4.1. 概要

整数値を扱うクラスです。

17.4.2. インスタンスの生成

整数を表す数値リテラルにサフィックス L をつけると gmp.mpz インスタンスを生成します。
また、以下のコンストラクタ関数を実行することでインスタンスを生成できます。

`gmp.mpz(value?) {block?}`

gmp.mpz インスタンスを生成します。引数 value には数値または数値として解釈できる文字列を渡します。
value を省略するとゼロ数値のインスタンスを生成します。
block を指定するとブロックパラメータ |num:gmp.mpz| を渡してその内容を評価します。この場合、戻り値はブロックの最後の評価値になります。

17.5. gmp.mpq クラス

17.5.1. 概要

分数値を扱うクラスです。

17.5.2. インスタンスの生成

任意の数値リテラルにサフィックス Lr をつけると gmp.mpq インスタンスを生成します。数値に小数が含まれる場合は小数部を分数値で表現した結果がその値になります。

また、以下のコンストラクタ関数を実行することでインスタンスを生成できます。

```
gmp.mpq(numer?, denom?:number) {block?}
```

gmp.mpq インスタンスを生成します。引数 numer および denom に数値を渡すと、各数値を分子と分母にしたインスタンスを返します。numer に文字列を渡すと、分数値として解釈した結果が値になります。両方の引数を省略するとゼロ数値のインスタンスを生成します。

block を指定するとブロックパラメータ |num:gmp.mpq| を渡してその内容を評価します。この場合、戻り値はブロックの最後の評価値になります。

17.5.3. インスタンスプロパティ

プロパティ	データ型	R/W	説明
denom	gmp.mpz	R	分母を返します
numer	gmp.mpz	R	分子を返します

17.6. gmp.mpf クラス

17.6.1. 概要

浮動小数点数値を扱うクラスです。

17.6.2. インスタンスの生成

小数点またはべき乗を表す文字 e または E を含む数値リテラルにサフィックス L をつけると gmp.mpf インスタンスを生成します。

また、以下のコンストラクタ関数を実行することでインスタンスを生成できます。

```
gmp.mpf(value?, prec?:number) {block?}
```

gmp.mpz インスタンスを生成します。引数 value には数値または数値として解釈できる文字列を渡します。value を省略するとゼロ数値のインスタンスを生成します。

引数 prec には内部表現のビット幅を指定します。省略すると gmp.mpf.set_default_prec() 関数で指定したビット幅になります。

block を指定するとブロックパラメータ |num:gmp.mpf| を渡してその内容を評価します。この場合、戻り値はブロックの最後の評価値になります。

17.6.3. インスタンスプロパティ

プロパティ	データ型	R/W	説明
prec	number	R/W	内部表現のビット幅を表します。このプロパティに数値を書き込むと、ビット幅を変更することができます。

17.6.4. クラスメソッド

`gmp.mpf.set_default_prec(prec:number):static:void`

gmp.mpf インスタンスを生成する際の内部表現ビット幅のデフォルト値を指定します。

`gmp.mpf.get_default_prec():static`

gmp.mpf インスタンスを生成する際の内部表現ビット幅のデフォルト値を取得します。

18. gurcbuild モジュール

18.1. 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。

18.2. サンプル

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。

スクリプト
import(gurcbuild) gurcbuild.build(['hoge.gura', 'image1.png', 'image2.png'])

18.3. 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。

19. gzip モジュール

19.1. 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/

19.2. サンプル

スクリプト

```
import(gzip)
gzip.writer('test.dat.gz').copyfrom('test.dat')
gzip.reader('test.dat.gz').copyto('test2.dat')
```

19.3. 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)

19.4. 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。

19.5. stream クラスの拡張

19.5.1. 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。

20. hash モジュール

20.1. 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。

20.2. サンプル

スクリプト

20.3. hash.hash クラス

20.3.1. インスタンスの生成

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

CRC32 値を算出する hash インスタンスを生成して返します。引数 stream を指定すると、インスタンス生成に続いてそのデータ内容をハッシュ内容に加えます。

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

MD5 値を算出する hash インスタンスを生成して返します。引数 stream を指定すると、インスタンス生成に続いてそのデータ内容をハッシュ内容に加えます。

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

SHA1 値を算出する hash インスタンスを生成して返します。引数 stream を指定すると、インスタンス生成に続いてそのデータ内容をハッシュ内容に加えます。

20.3.2. インスタンスプロパティ

プロパティ	データ型	R/W	説明
digest	binary	R	ハッシュ値をバイナリデータで返します
hexdigest	string	R	ハッシュ値を 16 進文字列にしたものを返します
number	number	R	ハッシュ値を数値にして返します。CRC32 のみ有効で、他のハッシュに対してこのプロパティを読むと nil を返します。

20.3.3. インスタンスメソッド

hash.hash#init():reduce

hash.hash#update(stream:stream:r):reduce

```
hash.hash#write(buff:binary, len?:number):reduce
```

21. http モジュール

21.1. 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/

21.2. サンプル

スクリプト

21.3. 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。

21.4. 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

21.5. 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。

21.6. http.server クラス

21.6.1. 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。

21.6.2. 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

21.6.3. 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。

21.6.4. 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。

スクリプト

```
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')
}
```

21.7. http.client クラス

21.7.1. 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。

21.7.2. 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。

21.7.3. 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'

21.7.4. 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。

スクリプト

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

21.8. http.stat クラス

21.8.1. 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。

21.8.2. 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。

21.8.3. 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

21.8.4. 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。

21.9. http.request クラス

21.9.1. 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。

21.9.2. 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。

21.9.3. 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.spliturl 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

21.9.4. 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。

21.10. http.session クラス

21.10.1. 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。

21.10.2. 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

21.11. http.response クラス

21.11.1. 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。

21.11.2. 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。

21.11.3. 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

21.11.4. 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。

22. jpeg モジュール

22.1. 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/

また、EXIF の実装は以下の URL の仕様に基づきます。

http://www.exif.org/specifications.html

22.2. サンプル

スクリプト

22.3. 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)

22.4. jpeg.exif クラス

22.4.1. 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。

22.4.2. 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。

22.4.3. 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。

22.4.4. 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。

22.5. jpeg.ifd クラス

22.5.1. 概要

Exif 内の IFD データの内容を表すクラスです。

22.5.2. インスタンスの生成

jpeg.exif インスタンスのプロパティ ifd0 および ifd1 のクラスとして使われます。

22.5.3. インスタンスプロパティ

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

そのほかに、IFD 内のタグデータに対応したプロパティが jpeg.tag クラスのインスタンスとして格納されています。タグデータの値は、各インスタンスの value プロパティで参照することができます。タグデータのプロパティ名と、各 value プロパティのデータ型を以下にまとめます。

TIFF Rev.6.0 Attribute Information - Tags relating to image data structure

プロパティ	データ型	R/W	内容
ImageWidth	number	R	
ImageLength	number	R	

BitsPerSample	number	R	
Compression	number	R	
PhotometricInterpretation	number	R	
Orientation	number	R	
SamplesPerPixel	number	R	
PlanarConfiguration	number	R	
YCbCrSubSampling	number	R	
YCbCrPositioning	number	R	
XResolution	fraction	R	
YResolution	fraction	R	
ResolutionUnit	number	R	

TIFF Rev.6.0 Attribute Information - Tags relating to recording offset

プロパティ	データ型	R/W	内容
StripOffsets	number	R	
RowsPerStrip	number	R	
StripByteCounts	number	R	
JPEGInterchangeFormat	number	R	
JPEGInterchangeFormatLength	number	R	

TIFF Rev.6.0 Attribute Information - Tags relating to image data characteristics

プロパティ	データ型	R/W	内容
TransferFunction	number	R	
WhitePoint	fraction	R	
PrimaryChromaticties	fraction	R	
YCbCrCoefficients	fraction	R	
ReferenceBlackWhite	fraction	R	

TIFF Rev.6.0 Attribute Information - Other tags

プロパティ	データ型	R/W	内容
DateTime	string	R	
ImageDescription	string	R	
Make	string	R	
Model	string	R	
Software	string	R	
Artist	string	R	
Copyright	string	R	

Exif IFD Attribute Information – Tags relating to Version

プロパティ	データ型	R/W	内容
ExifVersion	binary	R	
FlashPixVersion	binary	R	

Exif IFD Attribute Information – Tag relating to Image Data Characteristics

プロパティ	データ型	R/W	内容
ColorSpace	number	R	

Exif IFD Attribute Information – Tags relating to image configuration

プロパティ	データ型	R/W	内容
ComponentsConfiguration	binary	R	
CompressedBitsPerPixel	fraction	R	
PixelXDimension	number	R	
PixelYDimension	number	R	

Exif IFD Attribute Information – Tags relating to user information

プロパティ	データ型	R/W	内容
MakerNote	binary	R	
UserComment	binary	R	

Exif IFD Attribute Information – Tags relating to related file information

プロパティ	データ型	R/W	内容
RelatedSoundFile	string	R	

Exif IFD Attribute Information – Tags relating to date and time

プロパティ	データ型	R/W	内容
DateTimeOriginal	string	R	
DateTimeDigitized	string	R	
SubSecTime	string	R	
SubSecTimeOriginal	string	R	
SubSecTimeDigitized	string	R	

Exif IFD Attribute Information – Tags relating to picture-taking conditions

プロパティ	データ型	R/W	内容
ExposureTime	fraction	R	
FNumber	fraction	R	
ExposureProgram	number	R	
SpectralSensitivity	string	R	
ISOSpeedRatings	number	R	
OECF	binary	R	
ShutterSpeedValue	fraction	R	
ApertureValue	fraction	R	
BrightnessValue	fraction	R	
ExposureBiasValue	fraction	R	
MaxApertureValue	fraction	R	
SubjectDistance	fraction	R	
MeteringMode	number	R	

LightSource	number	R	
Flash	number	R	
FocalLength	fraction	R	
SubjectArea	number	R	
FlashEnergy	fraction	R	
SpatialFrequencyResponse	binary	R	
FocalPlaneXResolution	fraction	R	
FocalPlaneYResolution	fraction	R	
FocalPlaneResolutionUnit	number	R	
SubjectLocation	number	R	
ExposureIndex	fraction	R	
SensingMethod	number	R	
FileSource	binary	R	
SceneType	binary	R	
CFAPattern	binary	R	
CustomRendered	number	R	
ExposureMode	number	R	
WhiteBalance	number	R	
DigitalZoomRatio	fraction	R	
FocalLengthIn35mmFilm	number	R	
SceneCaptureType	number	R	
GainControl	fraction	R	
Contrast	number	R	
Saturation	number	R	
Sharpness	number	R	
DeviceSettingDescription	binary	R	
SubjectDistanceRange	number	R	

Exif IFD Attribute Information – Other Tags

プロパティ	データ型	R/W	内容
ImageUniqueID	string	R	

GPS Attribute Information

プロパティ	データ型	R/W	内容
GPSVersionID	binary	R	
GPSLatitudeRef	string	R	
GPSLatitude	fraction	R	
GPSLongitudeRef	string	R	
GPSLongitude	fraction	R	
GPSAltitudeRef	binary	R	

GPSAltitude	fraction	R	
GPSTimeStamp	fraction	R	
GPSSatellites	string	R	
GPSStatus	string	R	
GPSMeasureMode	string	R	
GPSDOP	fraction	R	
GPSSpeedRef	string	R	
GPSSpeed	fraction	R	
GPSTrackRef	string	R	
GPSTrack	fraction	R	
GPSImgDirectionRef	string	R	
GPSImgDirection	fraction	R	
GPSMapDatum	string	R	
GPSDestLatitudeRef	string	R	
GPSDestLatitude	fraction	R	
GPSDestLongitudeRef	string	R	
GPSDestLongitude	fraction	R	
GPSBearingRef	string	R	
GPSBearing	fraction	R	
GPSDestDistanceRef	string	R	
GPSDestDistance	fraction	R	
GPSProcessingMethod	binary	R	
GPSAreaInformation	binary	R	
GPSDateStamp	string	R	
GPSDifferential	number	R	

Interoperability IFD Attribute Information

プロパティ	データ型	R/W	内容
InteroperabilityIndex	string	R	
InteroperabilityVersion	binary	R	
RelatedImageWidth	number	R	
RelatedImageHeight	number	R	

22.5.4. インデクスアクセス

インデクサで要素名にタグシンボルあるいは文字列を指定することで、タグデータを取得することができます。

22.5.5. 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。

22.6. jpeg.tag クラス

22.6.1. 概要

タグデータの内容を表すクラスです。

22.6.2. インスタンスの生成

jpeg.ifd クラスのプロパティとして取得します。

22.6.3. 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	タグ ID
name	string	R	タグ名を文字列で返します。
symbol	symbol	R	タグ名をシンボル型で返します。
type	number	R	データ型
value	any	R	タグの値
ifd	jpeg.ifd	R	タグが他の IFD へのポインタになっている場合、その参照を返します。その他のデータ型の場合は nil を返します。

22.7. image クラスの拡張

22.7.1. 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。

23. markdown モジュール

23.1. 概要

markdown 文法（http://daringfireball.net/projects/markdown/）で記述されたドキュメントを解析するモジュールです。使用するには import 関数を使って markdown モジュールをインポートします。

23.2. サンプル

スクリプト
import(markdown) markdown.document('hoge.md').render_html('hoge.html')

23.3. モジュール構成

markdown モジュールはスクリプト形式のモジュールファイル markdown.gura とバイナリ形式のモジュールファイル markdown.gurd で構成されます。同じ名前のモジュールが存在した場合はスクリプト形式のモジュールファイルが優先してインポートされるので、import を実行すると初めに markdown.gura をインポートし、そこから markdown.gurd をインポートします。

markdown.gura には、HTML などへの変換方法が記述されています。

23.4. モジュール関数

markdown.setpresenter():void {block}

help() 関数などを呼んだときに実行する表示処理を block で登録します。

ブロックパラメータの形式は |title:string:nil, doc:markdown.document:nil| で、title にタイトル文字列、doc にヘルプ登録した markdown 形式のドキュメントを解析した結果を含んだ markdown.document インスタンスが入ります。タイトルがない場合、title には nil が入ります。ヘルプがない場合、doc には nil が入ります。

23.5. markdown.document クラス

23.5.1. インスタンスの生成

markdown.document(stream?:stream:r) {block?}

引数 stream で示されるストリームから markdown 文法のテキストを読み取り、解析した結果を含んだ markdown.document インスタンスを返します。引数 stream を省略した場合、内容が空の markdown.document インスタンスを返します。

23.5.2. インスタンスプロパティ

プロパティ	データ型	R/W	説明
root	markdown.item	R	
refs	iterator	R	

23.5.3. インスタンスメソッド

`markdown.document#parse(str:string):void`

文字列から markdown 文法のテキストを読み取り、現在の markdown.document インスタンスの内容に解析した結果を追加します。

`markdown.document#read(stream:stream:r):void`

引数 stream で示されるストリームから markdown 文法のテキストを読み取り、現在の markdown.document インスタンスの内容に解析した結果を追加します。

`markdown.document#render_html(out?:stream:w)`

内容を HTML フォーマットにして、引数 out で示すストリームに出力します。引数 out を省略すると、結果を文字列で返します。

`markdown.document#render_rtf(out?:stream:w)`

内容を RTF フォーマットにして、引数 out で示すストリームに出力します。引数 out を省略すると、結果を文字列で返します。

`markdown.document#render_console()`

内容をコンソールに出力します。

23.6. markdown.item クラス

23.6.1. アイテムタイプ

タイプ	説明
root	
h1	
h2	
h3	
h4	
h5	
h6	
p	
blockquote	
em	
strong	
codeblock	
ol	
ul	
li	
line	

a	
img	
text	
code	
entity	
tag	
hr	
br	
referee	

23.6.2. インスタンスプロパティ

プロパティ	データ型	R/W	説明
type	symbol	R	タイプシンボル
text	string	R	
children	iterator	R	子要素を markdown.item 型で返すイテレータ
url	string	R	
title	string	R	
attrs	string	R	

23.6.3. インスタンスメソッド

markdown.item#print():void

インスタンスの内容を表示します。子要素を持つ場合はそれらも再帰的に表示します。

24. math モジュール

24.1. 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。

24.2. サンプル

スクリプト

24.3. 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)`

与えられた制御点に基づいたベジエ曲線の関数を返します。

`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[])`

外積を計算します。

`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[])`

内積を計算します。

`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。

25. midi モジュール

25.1. 概要

MIDI 音源を制御してサウンドを出力したり、MIDI ファイルの読み書きを行うモジュールです。楽譜を MML で記述することができます。使用するには import 関数を使って midi モジュールをインポートします。

25.2. サンプル

25.2.1. MIDI ファイルを読み込んで演奏

スクリプト
import(midi) midi.port().play('Caccini_avemaria.mid')

25.2.2. MML を演奏

スクリプト
import(midi) midi.port().mml('CDEFGAB~C')

25.2.3. MML から MIDI ファイルを生成

スクリプト
import(midi) seq = midi.sequence() seq.mml('CDEFGAB~C') seq.write('simple.mid')

25.3. モジュールプロパティ

プロパティ	データ型	R/W	説明
programs	midi.program[]	R/W	

25.4. mml.event クラス

25.4.1. インスタンスの生成

25.4.2. インスタンスプロパティ

プロパティ	データ型	R/W	説明
type	symbol	R	
timestamp	number	R	
status	number	R	
name	string	R	
symbol	symbol	R	
args	string	R	

25.5. mml.track クラス

25.5.1. インスタンスの生成

25.5.2. インスタンスプロパティ

プロパティ	データ型	R/W	説明
events	iterator	R	トラック内に追加された midi.event インスタンスを要素に持つイテレータを返します

25.5.3. インスタンスメソッド

```
midi.track#seek(offset:number, origin?:symbol):reduce
```

トラック内のイベントの挿入位置を移動します。

```
midi.track#tell()
```

トラック内のイベントの挿入位置を返します。

```
midi.track#erase(n?:number):reduce
```

トラック内の指定の位置にあるイベントを削除します。

```
midi.track#mml(str:string, max_velocity?:number):map:reduce
```

MML 文字列をパースして track に追加します。

```
midi.track#note_off(channel:number, note:number,
                    velocity:number, deltaTime?:number):map:reduce
```

MIDI チャネルイベント "Note Off" をトラックに追加します。

```
midi.track#note_on(channel:number, note:number,
                    velocity:number, deltaTime?:number):map:reduce
```

MIDI チャネルイベント "Note On" をトラックに追加します。

```
midi.track#poly_pressure(channel:number, note:number,
                    value:number, deltaTime?:number):map:reduce
```

MIDI チャネルイベント "Poly Pressure" をトラックに追加します。

```
midi.track#control_change(channel:number, controller,
                    value:number, deltaTime?:number):map:reduce
```

MIDI チャネルイベント "Control Change" をトラックに追加します。

```
midi.track#program_change(channel:number,
                    program, deltaTime?:number):map:reduce
```

MIDI チャネルイベント "Program Change" をトラックに追加します。

```
midi.track#channel_pressure(channel:number,
                    pressure:number, deltaTime?:number):map:reduce
```

MIDI チャネルイベント "Channel Pressure" をトラックに追加します。

```
midi.track#pitch_bend(channel:number,
                      value:number, deltaTime?:number):map:reduce
```

MIDI チャネルイベント "Pitch Bend" をトラックに追加します。

```
midi.track#sequence_number(number:number, deltaTime?:number):map:reduce
```

MIDI メタイベント "Sequence Number" をトラックに追加します。

```
midi.track#text_event(text:string, deltaTime?:number):map:reduce
```

MIDI メタイベント "Text Event" をトラックに追加します。

```
midi.track#copyright_notice(text:string, deltaTime?:number):map:reduce
```

MIDI メタイベント "Copyright Notice" をトラックに追加します。

```
midi.track#sequence_or_track_name(text:string, deltaTime?:number):map:reduce
```

MIDI メタイベント "Sequence/Track Name" をトラックに追加します。

```
midi.track#instrument_name(text:string, deltaTime?:number):map:reduce
```

MIDI メタイベント "Instrument Name" をトラックに追加します。

```
midi.track#lyric_text(text:string, deltaTime?:number):map:reduce
```

MIDI メタイベント "Lyric Text" をトラックに追加します。

```
midi.track#marker_text(text:string, deltaTime?:number):map:reduce
```

MIDI メタイベント "Marker Text" をトラックに追加します。

```
midi.track#cue_point(text:string, deltaTime?:number):map:reduce
```

MIDI メタイベント "Cue Point" をトラックに追加します。

```
midi.track#midi_channel_prefix_assignment(channel:number,
                                          deltaTime?:number):map:reduce
```

MIDI メタイベント "Channel Prefix Assignment" をトラックに追加します。

```
midi.track#end_of_track(deltaTime?:number):map:reduce
```

MIDI メタイベント "" をトラックに追加します。

```
midi.track#tempo_setting(mpqn:number, deltaTime?:number):map:reduce
```

MIDI メタイベント "Tempo Setting" をトラックに追加します。

```
midi.track#smpte_offset(hour:number, minute:number, second:number,
              frame:number, subFrame:number, deltaTime?:number):map:reduce
```

MIDI メタイベント "SMPTE Offset" をトラックに追加します。

```
midi.track#time_signature(numerator:number, denominator:number,
          metronome:number, cnt32nd:number, deltaTime?:number):map:reduce
```

MIDI メタイベント "Time Signature" をトラックに追加します。

```
midi.track#key_signature(key:number,
                         scale:number, deltaTime?:number):map:reduce
```

MIDI メタイベント "Key Signature" をトラックに追加します。

```
midi.track#sequencer_specific_event(binary:binary,
                                    deltaTime?:number):map:reduce
```

MIDI メタイベント "Sequencer Specific Event" をトラックに追加します。

25.6. mml.sequence クラス

25.6.1. インスタンスの生成

25.6.2. インスタンスプロパティ

プロパティ	データ型	R/W	説明
format	number	R/W	
tracks	iterator	R	シーケンス内の midi.track インスタンスを要素に持つイテレータを返します
events	iterator	R	
division	number	R/W	

25.6.3. インスタンスメソッド

```
midi.sequence#read(stream:stream:r):map:reduce
```

ストリームからスタンダード MIDI ファイルを読み込みます。

```
midi.sequence#write(stream:stream:w):map:reduce
```

シーケンスの内容を標準ストリームにスタンダード MIDI ファイル形式で書き出します。

```
midi.sequence#play(port:midi.port,
                   speed?:number, repeat:number:nil => 1):[background,player]
```

指定の MIDI ポートでシーケンスの内容を演奏します。

```
midi.sequence#track(index:number):map {block?}
```

インデクスで指定された track インスタンスが sequence にある場合はそれを返します。存在しない場合は、新しくインスタンスを生成して返します。その際、すでにあるトラックのインデクス値と新たなインデクス値の間があいている場合、中間の track インスタンスも生成します。

```
midi.sequence#mml(str:string, max_velocity?:number):reduce
```

MML 文字列をパースして sequence に追加します。

```
midi.sequence#readmml(stream:stream, max_velocity?:number):reduce
```

ストリームから MML 文字列を読み込んでパースし、sequence に追加します。

25.7. midi.portinfo クラス

25.7.1. インスタンスの生成

25.7.2. インスタンスプロパティ

プロパティ	データ型	R/W	説明
		R/W	

25.7.3. インスタンスメソッド

25.8. midi.port クラス

25.8.1. インスタンスの生成

25.8.2. インスタンスメソッド

```
midi.port#send(msg+:number):map:reduce
```
MIDI イベントメッセージをポートに送信します。

```
midi.port#play(sequence:midi.sequence, speed?:number,
                            repeat:number:nil => 1):map:[background,player]
```

```
midi.port#mml(str:string, max_velocity?:number):[background,player]
```
MML 文字列をパースして得た MIDI チャネルイベントをポートに送信します。

```
midi.port#readmml(stream:stream, max_velocity?:number):[background,player]
```
ストリームから MML 文字列を読み込んでパースして得た MIDI チャネルイベントをポートに送信します。

```
midi.port#note_off(channel:number, note:number, velocity:number):map:reduce
```
MIDI チャネルイベント "Note Off" をポートに送信します。

```
midi.port#note_on(channel:number, note:number, velocity:number):map:reduce
```
MIDI チャネルイベント "Note On" をポートに送信します。

```
midi.port#poly_pressure(channel:number, note:number, value:number):map:reduce
```
MIDI チャネルイベント "Poly Pressure" をポートに送信します。

```
midi.port#control_change(channel:number,
                            controller:number, value:number):map:reduce
```
MIDI チャネルイベント "Control Change" をポートに送信します。

```
midi.port#program_change(channel:number, program:number):map:reduce
```
MIDI チャネルイベント "Program Change" をポートに送信します。

`midi.port#channel_pressure(channel:number, pressure:number):map:reduce`

MIDI チャネルイベント "Channel Pressure" をポートに送信します。

`midi.port#pitch_bend(channel:number, value:number):map:reduce`

MIDI チャネルイベント "Pitch Bend" をポートに送信します。

25.9. `midi.player` クラス

25.9.1. インスタンスの生成

25.9.2. インスタンスプロパティ

プロパティ	データ型	R/W	説明
speed	number	R/W	
count	number	R	
repeat	number	R	
progress	number	R	

25.9.3. インスタンスメソッド

25.10. `midi.controller` クラス

25.10.1. インスタンスの生成

25.10.2. クラスプロパティ

プロパティ	データ型	R/W	説明
	midi.controller	R/W	

25.10.3. インスタンスプロパティ

プロパティ	データ型	R/W	説明
		R/W	

25.10.4. インスタンスメソッド

25.11. `midi.program` クラス

25.11.1. インスタンスの生成

25.11.2. クラスプロパティ

プロパティ	データ型	R/W	説明
	midi.program	R/W	

25.11.3. インスタンスプロパティ

プロパティ	データ型	R/W	説明
		R/W	

25.11.4. インスタンスメソッド

25.12. midi.soundfont クラス

25.12.1. インスタンスの生成

25.12.2. インスタンスプロパティ

プロパティ	データ型	R/W	説明
		R/W	

25.12.3. インスタンスメソッド

```
midi.soundfont#synthesizer(preset:number,      bank:number,      key:number,
    velocity:number):map {block?}
```

25.13. midi.synthesizer クラス

25.13.1. インスタンスの生成

25.13.2. インスタンスメソッド

25.14. MML 文法

音程は A, B, C, D, E, F, G の文字で表します。音程文字の後に # または + をつけると半音あがります。音程文字の後に - をつけると半音下がります。

音程の後に数値をつけると長さを指定することができます。たとえば、"C4" はドの四分音符を意味します。

長さ数値の後にドット記号をつけると、付点音符になります。たとえば、"C4." はドの付点四分音符を意味します。

音符の長さを指定しないと、デフォルトの長さになります。この長さは、初期化時は四分音符になりますが、L オペレータでデフォルト長を変更することができます。"L8" はデフォルト長を四分音符にします。

音符の長さ指定の後に、カンマに続いて 0 から 8 までの数値を記述すると、実際に音を発声するゲート時間を指定できます。たとえば、"C4,1" は四分音符ですが発声時間はその長さの 1/8 になります。これはスタッカートを表現するときに使われます。ゲート時間を指定しないときは 8 になりますが、Q*n* でこのデフォルト値を変更することができます。

休符はオペレータ R で記述します。長さ指定の方法は音符と同じです。長さを指定しない場合はオペレータ L

で指定したデフォルト長になります。

音程文字の前に記号 ~ をつけるとオクターブが一つあがり、記号 _ をつけると一つ下がります。オクターブを複数段階上げ下げする場合は、これらの記号をその回数だけ繰り返します。このオクターブ指定は直後の音符に対してのみ有効です。

以降のオクターブをすべて変更したい場合はオペレータ < および > を使います。オペレータ < でオクターブを上げ、> で下げます。オペレータ o に続いてオクターブのレベルを数値で指定することもできます。オクターブレベルは 1 から 9 までの数値で、初期化時は 4 になります。

和音を記述するときは、音符どうしをコロン記号でつなげます。四分音符でドミソの和音を鳴らす場合は "C4:E4:G4" とします。

複数の音符を括弧でくくって長さ数値を指定すると、括弧内の音符の発声時間がその時間に調整されます。たとえば、"(CDE)4" と記述すると、これは全体の長さが四分音符の連譜になります。括弧の中身を和音にすると、長さ指定を一括して行うことができます。四分音符でドミソの和音を鳴らす場合は "(C:E:G)4" と記述できます。

音符列を "[*part*]*n*" のようにブラケットでくくると繰り返しになります。*part* は繰り返す音符列、*n* は繰り返す回数です。*n* を省略すると 2 回繰り返します。ブラケットの内容を "[*part*|*last*]*n*" のように記述すると、| の後に続く部分は最後の繰り返しでは演奏されません。たとえば "[CDE|FG]3" は CDEFGCDEFGCDE を演奏します。音符列の中に繰り返しを記述することもできます。

音符同士をつなげる場合は & を使います。同じ音程の音符をこれで結合すると、つなげた長さぶんだけ切れ目なしに演奏します。音程が異なる場合は、スラーの表現になりますが、現在はサポートされていません。

ベロシティを指定するには V*n* と記述します。*n* は 0 から 127 までの数値です。

音色を指定するには @*n* と記述します。*n* は GM 規格で定められた音色番号から 1 を引いたもので、0 から 127 までの数値になります。たとえば、グランドピアノの場合 0、トランペットは 56 を指定します。

音色を名前で指定することもでき、その場合は @{*name*} のように記述します。音色の名前として指定できるものを以下にまとめます。

n	音色名	n	音色名
0	acoustic_piano	32	acoustic_bass
1	bright_piano	33	electric_bass_finger
2	electric_grand_piano	34	electric_bass_pick
3	honky_tonk_piano	35	fretless_bass
4	electric_piano	36	slap_bass_1
5	electric_piano_2	37	slap_bass_2
6	harpsichord	38	synth_bass_1
7	clavi	39	synth_bass_2
8	celesta	40	violin
9	glockenspiel	41	viola
10	musical_box	42	cello
11	vibraphone	43	double_bass
12	marimba	44	tremolo_strings

13	xylophone	45	pizzicato_strings
14	tubular_bell	46	orchestral_harp
15	dulcimer	47	timpani
16	drawbar_organ	48	string_ensemble_1
17	percussive_organ	49	string_ensemble_2
18	rock_organ	50	synth_strings_1
19	church_organ	51	synth_strings_2
20	reed_organ	52	voice_aahs
21	accordion	53	voice_oohs
22	harmonica	54	synth_voice
23	tango_accordion	55	orchestra_hit
24	acoustic_guitar_nylon	56	trumpet
25	acoustic_guitar_steel	57	trombone
26	electric_guitar_jazz	58	tuba
27	electric_guitar_clean	59	muted_trumpet
28	electric_guitar_muted	60	french_horn
29	overdriven_guitar	61	brass_section
30	distortion_guitar	62	synth_brass_1
31	guitar_harmonics	63	synth_brass_2

n	音色名	n	音色名
64	soprano_sax	96	fx_1_rain
65	alto_sax	97	fx_2_soundtrack
66	tenor_sax	98	fx_3_crystal
67	baritone_sax	99	fx_4_atmosphere
68	oboe	100	fx_5_brightness
69	english_horn	101	fx_6_goblins
70	bassoon	102	fx_7_echoes
71	clarinet	103	fx_8_sci_fi
72	piccolo	104	sitar
73	flute	105	banjo
74	recorder	106	shamisen
75	pan_flute	107	koto
76	blown_bottle	108	kalimba
77	shakuhachi	109	bagpipe
78	whistle	110	fiddle
79	ocarina	111	shanai
80	lead_1_square	112	tinkle_bell

81	lead_2_sawtooth	113	agogo
82	lead_3_calliope	114	steel_drums
83	lead_4_chiff	115	woodblock
84	lead_5_charang	116	taiko_drum
85	lead_6_voice	117	melodic_tom
86	lead_7_fifths	118	synth_drum
87	lead_8_bass_lead	119	reverse_cymbal
88	pad_1_fantasia	120	guitar_fret_noise
89	pad_2_warm	121	breath_noise
90	pad_3_polysynth	122	seashore
91	pad_4_choir	123	bird_tweet
92	pad_5_bowed	124	telephone_ring
93	pad_6_metallic	125	helicopter
94	pad_7_halo	126	applause
95	pad_8_sweep	127	gunshot

テンポ指定はオペレータ `T`*n* で行います。*n* は 1 分間に演奏する四分音符の数です。

ラインコメントは `//` で始まります。ブロックコメントは `/*` から `*/` までの間になります。

26. modbuild モジュール

26.1. 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。

26.2. サンプル

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。

スクリプト

`import(modbuild)`
`builder = modbuild.Builder()`
`builder.build('hoge', ['module-hoge.cpp'])`

26.3. modbuild.Builder クラス

26.4. インスタンスプロパティ

プロパティ	データ型	R/W	説明
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

26.5. 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。

27. modgen モジュール

27.1. 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。

27.2. 使い方

このモジュールはコマンドラインから実行されます。たとえばバイナリモジュール hoge のひな型を作る場合は以下のように gura インタープリターを実行します。

```
$ gura -i modgen hoge
```

28. msico モジュール

28.1. 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

28.2. サンプル

スクリプト

28.3. 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）

28.4. msico.content クラス

28.4.1. 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。

28.4.2. 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。

28.4.3. 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。

28.5. `image` クラスの拡張

28.5.1. 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。

29. mswin モジュール

29.1. 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。

29.2. サンプル

スクリプト

29.3. mswin.ole クラス

29.3.1. 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。

29.4. mswin.regkey クラス

29.4.1. 概要

Windows のレジストリを扱うクラスです。

29.4.2. 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
```

29.4.3. 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。

29.5. COM について

29.5.1. 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。

スクリプト

```
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。

スクリプト

```
import(mswin)
mswin.ole('Word.Application') {|app|
    app.Visible = 1
    app.Documents.Open(path.absname('hoge.doc'))
}
```

29.5.2. 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。

29.5.3. 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。

29.5.4. 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。

29.5.5. 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。

スクリプト

```
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。

30. opengl モジュール

「Gura モジュールリファレンス – opengl」を参照ください。

31. os モジュール

31.1. 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。

31.2. サンプル

スクリプト

31.3. 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。

31.4. モジュール関数

os.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。

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.sleep(secs:number)

指定した秒数だけスリープします。

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。

32. path モジュール

32.1. 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。

32.2. サンプル

スクリプト

32.3. 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、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。

33. png モジュール

33.1. 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

33.2. サンプル

スクリプト

33.3. 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)

33.4. image クラスの拡張

33.4.1. 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。

34. ppm モジュール

34.1. 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 に掲載されている仕様に基づきます。

http://local.wasp.uwa.edu.au/~pbourke/dataformats/ppm/

34.2. サンプル

スクリプト

34.3. 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）

34.4. image クラスの拡張

34.4.1. 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。

35. re モジュール

35.1. 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

35.2. サンプル

スクリプト

35.3. 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。

35.4. 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(m:re.match)
```

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。

35.5. re.match クラス

35.5.1. 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

35.5.2. 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。

35.5.3. 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

35.5.4. 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。

35.6. re.pattern クラス

35.6.1. 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。

35.6.2. 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。

35.7. string クラスの拡張

35.7.1. 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、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。

35.8. list/iterator クラスの拡張

35.8.1. 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+')
```

36. sed モジュール

36.1. 概要

ファイル中の文字列を置換するモジュールです。使用するには import 関数を使って sed モジュールをインポートします。

36.2. サンプル

スクリプト

36.3. モジュール関数

```
sed.glob(pattern:string):[silent] {`block}

sed.walk(directory?:directory, maxdepth?:number, pattern*:string):
                                                    [silent] {`block}
```

37. sdl モジュール

「Gura モジュールリファレンス – sdl」を参照ください。

38. sqlite3 モジュール

38.1. 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

38.2. サンプル

スクリプト

38.3. 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

38.4. sqlite3.db クラス

38.4.1. 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。

38.4.2. 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。

39. sys モジュール

39.1. 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。

39.2. サンプル

スクリプト

39.3. 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。

39.4. 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

40. tar モジュール

40.1. 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。

40.2. サンプル

スクリプト

40.3. 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。

40.4. 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

40.5. tar.reader クラス

40.5.1. 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

40.5.2. 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。

40.6. tar.writer クラス

40.6.1. 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

40.6.2. 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。

40.7. `tar.stat` クラス

40.7.1. 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

41. `tcl` モジュール

「Gura モジュールリファレンス – tcl, tk」を参照ください。

42. tk モジュール

「Gura モジュールリファレンス – tcl, tk」を参照ください。

43. tokenizer モジュール

t.b.d

44. utils モジュール

44.1. 概要

ユーティリティ関数やクラスをまとめたモジュールです。

44.2. Aligner クラス

44.2.1. 概要

複数の文字列に対してタブ文字を付け加えて右端の位置をそろえます。

44.2.2. サンプル

スクリプト

```
import(utils)
lines = [
    '0'
    '01'
    '012'
    '0123'
    '01234'
    '012345'
    '0123456'
    '01234567'
    '012345678'
    '0123456789'
]
aligner = utils.Aligner(lines)
println(lines, aligner.tab(lines), '.. alinged')
```

44.2.3. インスタンスの生成

`utils.Aligner(strs[]:string, wdTab => 4) {block?}`

文字列配列 strs のアラインメントをそろえる Aligener インスタンスを生成します。引数 wdTab にタブ文字の文字幅を指定します。

block を指定すると、ブロックパラメータを |aligner:Aligner| という形式で渡してその内容を評価します。このとき、ブロックの最後の評価値が関数の戻り値になります。

44.2.4. インスタンスメソッド

`utils.Aligner#tab(str:string):map`

文字列 str のアラインメントをそろえるタブ文字列を生成します。

45. units モジュール

t.b.d

46. uuid モジュール

46.1. 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。

46.2. サンプル

スクリプト

46.3. 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。

47. wx モジュール

「Gura モジュールリファレンス – wx」を参照ください。

48. xml モジュール

48.1. 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/

48.2. サンプル

スクリプト

48.3. モジュール関数

```
xml.read(stream:stream:r)
```

48.4. xml.parser クラス

48.4.1. 使用例

スクリプト

```
import(xml)
Parser = class(xml.parser) {
    StartElement(elem:xml.element) = {
        printf('%s¥n', elem)
    }
    EndElement(tagname:string) = {
        println(name)
    }
    CharacterData(text:string) = {
        print(text)
    }
}
```

48.4.2. インスタンスの生成

```
xml.parser()
```

48.4.3. 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()
```

48.4.4. インスタンスメソッド

```
xml.parser#parse(stream:stream)
```

48.5. xml.attribute クラス

48.5.1. インスタンスプロパティ

プロパティ	データ型	R/W	説明
name	string	R	
value	string	R	

48.6. xml.element クラス

48.6.1. インスタンスの生成

```
xml.element(_tagname_:string, attrs%) {block?}
```

```
xml.comment(comment:string)
```

48.6.2. インスタンスプロパティ

プロパティ	データ型	R/W	説明
tagname	string	R	
text	string	R	
comment	string	R	
children	iterator	R	xml.element
attrs	iterator	R	xml.attribute

48.6.3. インスタンスメソッド

```
xml.element#gendoc(stream?:stream:w, fancy?:boolean, indentLevel?:number)
```

```
xml.element#gettext()
```

```
xml.element#addchild(value):void:map
```

48.6.4. オペレータ

```
xml.element << any
```

48.7. xml.document クラス

48.7.1. インスタンスの生成

```
xml.document(stream?:stream:r) {block?}
```

48.7.2. インスタンスプロパティ

プロパティ	データ型	R/W	説明
version	string	R	
encoding	string	R	
root	xml.element	R/W	

48.7.3. インスタンスメソッド

```
xml.document#gendoc(stream?:stream:w, fancy?:boolean)
```

48.8. stream クラスの拡張

48.8.1. インスタンスメソッド

```
stream#xmlread()
```

49. xpm モジュール

49.1. 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。

49.2. サンプル

スクリプト

49.3. 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）

49.4. image クラスの拡張

49.4.1. 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。

50. yaml モジュール

50.1. 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/

50.2. サンプル

スクリプト

50.3. データオブジェクトの対応

YAML 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。

YAML	スクリプト
sequence	list
mapping	dict
scalar	string はそのまま。その他のデータは string に変換

50.4. 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。

50.5. stream クラスの拡張

50.5.1. 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。

51. zip モジュール

51.1. 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

51.2. サンプル

スクリプト

51.3. 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。

51.4. zip.reader クラス

51.4.1. 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。

51.4.2. 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。

51.5. zip.writer クラス

51.5.1. 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

51.5.2. 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。

51.6. zip.stat クラス

51.6.1. 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