Gura ライブラリリファレンス

Updated: March 11, 2011

目次

1. この文書について 7
2. 組込み関数 7
2.1. インスタンス生成関数 7
2.2. 制御構文 8
2.2.1. 繰り返し 8
2.2.2. 条件分岐 8
2.2.3. 例外処理 9
2.2.4. switch 文 9
2.3. データ変換 9
2.4. クラス操作 10
2.5. ストリーム操作 10
2.6. 変数スコープ操作 10
2.7. イテレータ生成 10
2.8. フォーマット変換 11
2.9. モジュール 11
2.10. データ型チェック 11
2.11. 演算・統計 12
2.12. テキスト表示 12
2.13. スクリプト評価 13
2.14. 乱数 13
2.15. その他 13
3. 組込みクラス 13
3.1. binary クラス 13
3.1.1. インスタンスの生成 13
3.1.2. メソッドリファレンス 13
3.2. codec クラス 14
3.2.1. インスタンスの生成 14
3.2.2. メソッドリファレンス 14
3.3. color クラス 14
3.3.1. インスタンスの生成 14
3.3.2. メンバ変数 14
3.3.3. メソッドリファレンス 15
3.4. datetime クラス 15
3.4.1. インスタンスの生成 15
3.4.2. メソッドリファレンス 15
3.5. dict クラス 15

3.5.1. インスタンスの生成 15
3.5.2. メソッドリファレンス 15
3.6. environment クラス 16
3.6.1. インスタンスの生成 16
3.6.2. メソッドリファレンス 16
3.7. error クラス 17
3.7.1. インスタンスの生成 17
3.7.2. メソッドリファレンス 17
3.8. expr クラス 17
3.8.1. インスタンスの生成 17
3.8.2. メソッドリファレンス 17
3.9. function クラス 19
3.9.1. インスタンスの生成 19
3.9.2. メソッドリファレンス 19
3.10. image クラス 20
3.10.1. インスタンスの生成 20
3.11. メソッドリファレンス 20
3.12. iterator クラス 21
3.12.1. インスタンスの生成 21
3.12.2. メソッドリファレンス 21
3.13. list クラス 22
3.13.1. インスタンスの生成 22
3.13.2. メソッドリファレンス 22
3.14. matrix クラス 26
3.14.1. インスタンスの生成 26
3.14.2. メソッドリファレンス 26
3.15. palette クラス 27
3.15.1. インスタンスの生成 27
3.15.2. メソッドリファレンス 27
3.16. semaphore クラス 27
3.16.1. インスタンスの生成 27
3.16.2. メソッドリファレンス 27
3.17. stream クラス 27
3.17.1. インスタンスの生成 27
3.17.2. メソッドリファレンス 28
3.18. string クラス 29
3.18.1. インスタンスの生成 29
3.18.2. メソッドリファレンス 29
3.19. timedelta クラス 30

3.19.1. インスタンスの生成 30
3.19.2. メソッドリファレンス 30
4. sys モジュール 31
4.1. 関数 31
4.2. 変数 31
5. fs モジュール 31
5.1. 関数 32
5.2. fs.stat クラス 32
6. os モジュール 32
6.1. 関数 33
7. path モジュール 33
7.1. 関数 33
8. math モジュール 34
8.1. 関数 34
9. string モジュール 36
9.1. 関数 36
10. time モジュール 36
10.1. 関数 36
11. hash モジュール 37
11.1. 関数 37
12. ネットワーク – net.http モジュール 37
12.1. 関数 37
13. イメージファイル – bmp モジュール 38
14. イメージファイル – png モジュール 38
15. イメージファイル – jpeg モジュール 38
16. イメージファイル – msicon モジュール 38
17. イメージファイル – ppm モジュール 38
18. イメージファイル – gif モジュール 38
19. 二次元イメージ描画 – cairo モジュール 40
19.1. 命名規則 40
19.2. 関数 41
19.3. cairo.surface クラス 41
19.4. cairo.pattern クラス 42
19.5. cairo.region クラス 42
19.6. cairo.context クラス 43
20. canvas モジュール 47
21. フォント描画 – freetype モジュール 47
22. 三次元描画 – opengl および glu モジュール 47
22.1. 命名規則 47

23. 高速画面描画 – sdl モジュール 47
23.1. 命名規則 47
23.2. sdl.Cursor クラス 47
23.3. sdl.Timer クラス 47
23.4. sdl.PixelFormat クラス 47
23.5. sdl.PixelFormat クラス 47
23.6. sdl.Overlay クラス 48
23.7. sdl.Joystick クラス 48
23.8. sdl.AudioSpec クラス 48
23.9. sdl.AudioCVT クラス 48
23.10. sdl.CD クラス 49
24. グラフィカルユーザインターフェース – tcl および tk モジュール 49
24.1. モジュール構成 49
24.2. 命名規則 49
24.3. イベントの扱い 49
24.4. モジュール関数 50
24.5. image クラスへの追加メソッド 51
24.6. tk.Widget クラス 51
24.6.1. ウィジェット生成メソッド 54
24.7. tk.Menu クラス 56
24.8. tk.Canvas クラス 56
24.9. tk.CanvasItem クラス 56
24.10. tk.CanvasTag クラス 56
24.11. tk.Text クラス 56
24.12. tk.TextTag クラス 56
24.13. tk.Notebook クラス 56
24.14. tk.Treeview クラス 56
24.15. tk.TreeviewItem クラス 56
24.16. tk.TreeviewTag クラス 56
24.17. tk.FontT クラス 56
24.18. tk.Image クラス 56
24.19. 注意事項 56
25. データベース – sqlite3 モジュール 57
25.1. 関数 57
25.2. sqlite3 クラス 57
26. オーディオ 57
26.1. wav 57
27. アーカイブファイル – gzip モジュール 57
28. アーカイブファイル – bzip2 モジュール 57

29. アーカイブファイル – zipfile モジュール 58
29.1. 関数 58
29.2. zipfile.zipfile クラス 58
29.3. zipfile.stat クラス 58
30. アーカイブファイル – tar モジュール 59
30.1. 関数 59
30.2. tar.tar クラス 59
30.3. tar.stat クラス 59
31. テキスト処理 – re モジュール 60
31.1. 関数 60
31.2. re.match クラス 61
31.3. re.pattern クラス 61
31.4. string クラスへの追加メソッド 61
32. テキスト処理 – csv モジュール 61
32.1. 関数 62
32.2. stream クラスへの追加メソッド 62
33. テキスト処理 – xml モジュール 62
33.1. 関数 62
33.2. stream クラスへの追加メソッド 62
34. テキスト処理 – yaml モジュール 62
34.1. 関数 62
34.2. ストリームクラスへの追加メソッド 63
35. プラットフォーム – win32 モジュール 63
35.1. win32.regkey クラス 63
36. プラットフォーム – uuid モジュール 64
37. 開発ツール 64
37.1. lets_module 64
37.2. module_builder 64

1. この文書について

スクリプト言語 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。内容は Gura v0.1.0 の実装に基づきます。

2. 組込み関数

2.1. インスタンス生成関数

`binary(buff*)`

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

`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。

`color(args+):map`

指定した RGB 値または名前に対応する `color` 型インスタンスを返します。以下の呼び出し形式があります。

`color(name, alpha?:number)`

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)

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。

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

`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。

`image(args+):map {block?}`

```
lambda(`args*) {block} / &{block}
```

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

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

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

```
object()
```

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

```
palette(type:symbol)
```

```
semaphore()
```

2.2. 制御構文

2.2.1. 繰り返し

```
cross (`expr+) {block}
```

```
for (`expr+) {block}
```

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

```
while (`cond) {block}
```

2.2.2. 条件分岐

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

```
elsif (`cond) {block}
```

```
else() {block}
```

2.2.3. 例外処理

```
try() {block}
```

```
except(errors*:error) {block}
```

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

2.2.4. switch 文

```
switch() {block}
```

```
case(`cond) {block}
```

```
default() {block}
```

2.3. データ変換

```
chr(num:number):map
```

```
int(value):map
```

```
ord(str:string):map
```

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

```
tostring(value):map
```

```
tosymbol(str:string):map
```

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

2.4. クラス操作

```
class(superclass?:function):[static] {block?}
```

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

```
classref(type:expr):map
```

2.5. ストリーム操作

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

```
open(name:string, mode:string => "r", encoding:string => "utf-8"):map {block?}
```

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

2.6. 変数スコープ操作

```
scope(module?:module) {block}
```

```
extern(`syms+)
```

```
local(`syms+)
```

```
locals(module?:module)
```

```
outers()
```

```
typename(`value)
```

```
undef(`value+):[raise]
```

2.7. イテレータ生成

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

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

```
iterator(value+)
```

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

2.8. フォーマット変換

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

```
zip(values+) {block?}
```

2.9. モジュール

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

```
module() {block}
```

2.10. データ型チェック

関数	説明
isbinary(value)	
isboolean(value)	
isclass(value)	
iscomplex(value)	
isdatetime(value)	
isdict(value)	
isenvironment(value)	
iserror(value)	
isexpr(value)	
isfunction(value)	
isiterator(value)	
islist(value)	
ismatrix(value)	
ismodule(value)	

isnumber(value)	
issemaphore(value)	
isstring(value)	
issymbol(value)	
istimedelta(value)	
isuri(value)	

```
isdefined(`symbol)
```

```
isinstance(value, type:expr):map
```

```
istype(value, type:expr):map
```

2.11. 演算・統計

```
choose(index:number, values+):map
```

```
chooseif(flag:boolean, value1, value2):map
```

```
max(values+):map
```

```
min(values+):map
```

```
mod(n, m):map
```

```
in(n, m)
```

2.12. テキスト表示

```
print(value*):map:void
```

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

```
println(value*):map:void
```

2.13. スクリプト評価

```
eval(expr:expr):map
```

```
parse(code:string):map
```

```
parsestream(stream:stream):map
```

2.14. 乱数

```
rand(range?:number)
```

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

2.15. その他

```
dir(obj?)
```

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

3. 組込みクラス

3.1. binary クラス

3.1.1. インスタンスの生成

- コード中にバイナリリテラルを記述すると、`binary` インスタンスの生成になります。
- 関数 `pack` でバイナリ列のフォーマット指定をし、数値や文字列を埋め込んだインスタンスを生成することができます。

3.1.2. メソッドリファレンス

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

```
binary#decode(codec:codec)
```

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

```
binary#each() {block?}

binary#len()

binary#pointer(offset:number => 0)

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

binary#stream()

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

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

3.2. codec クラス

3.2.1. インスタンスの生成

- 関数 codec でインスタンスを生成します。

3.2.2. メソッドリファレンス

```
codec#decode(buff:binary):map

codec#encode(string:string):map
```

3.3. color クラス

3.3.1. インスタンスの生成

- 関数 color でインスタンスを生成します。

3.3.2. メンバ変数

変数名	読み書き	説明
red	R/W	赤要素を 0 から 255 までの数値で表します
green	R/W	緑要素を 0 から 255 までの数値で表します
blue	R/W	青要素を 0 から 255 までの数値で表します

alpha	R/W	アルファ要素を 0 から 255 までの数値で表します
gray	R	グレー値を取得します

3.3.3. メソッドリファレンス

```
color#html()
```

```
color#tolist(alpha:boolean => false)
```

3.4. datetime クラス

3.4.1. インスタンスの生成

- 関数 time.datetime で、年月日および時刻を指定したインスタンスを生成します。
- 関数 time.time で、時刻を指定したインスタンスを生成します。
- 関数 time.today で、今日の日付が入ったインスタンスを生成します。
- 関数 time.now で、現在の年月日および時刻が入ったインスタンスを生成します。

3.4.2. メソッドリファレンス

```
datetime#clrtzoff():reduce
```

```
datetime#format(format)
```

```
datetime#settzoff(mins:number):reduce
```

```
datetime#utc()
```

3.5. dict クラス

3.5.1. インスタンスの生成

- 辞書宣言 %{…} でインスタンスを生成します。
- 関数 dict でインスタンスを生成します。

3.5.2. メソッドリファレンス

```
dict#clear()
```

```
dict#erase(key):map
```

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

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

dict#haskey(key):map

dict#items() {block?}

dict#keys() {block?}

dict#len()

dict#set(key, value:nomap):map:void

dict#setdefault(key, default)

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

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

dict#values() {block?}
```

3.6. environment クラス

3.6.1. インスタンスの生成

3.6.2. メソッドリファレンス

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

3.7. error クラス

3.7.1. インスタンスの生成

3.7.2. メソッドリファレンス

3.8. expr クラス

3.8.1. インスタンスの生成

3.8.2. メソッドリファレンス

```
expr#block()
```

```
expr#car()
```

```
expr#cdr()
```

```
expr#child()
```

```
expr#each() {block?}
```

```
expr#exprname()
```

```
expr#getstring()
```

```
expr#getsymbol()
```

```
expr#getvalue()
```

```
expr#isassign()
```

expr#isbinary()

expr#isbinaryop()

expr#isblock()

expr#isblockparam()

expr#iscaller()

expr#iscontainer()

expr#isdictassign()

expr#isfield()

expr#isforce()

expr#isindexer()

expr#islister()

expr#isprefix()

expr#isquote()

expr#isstring()

expr#issuffix()

expr#issymbol()

```
expr#isunary()
```

```
expr#isunaryop()
```

```
expr#isvalue()
```

```
expr#left()
```

```
expr#right()
```

```
expr#tofunction(`args*)
```

```
expr#unquote()
```

3.9. function クラス

3.9.1. インスタンスの生成

- 関数を定義すると、function インスタンスが生成されます。
- 関数 lambda でインスタンスを生成します。

3.9.2. メソッドリファレンス

```
function#argsymbols()
```

```
function#expr()
```

```
function#fullname()
```

```
function#help(help?:string)
```

```
function#name()
```

```
function#setsymbol(symbol:symbol):reduce
```

3.10. image クラス

3.10.1. インスタンスの生成

- 関数 image でインスタンスを生成します。
- イメージ操作に関わるモジュールに、image インスタンスを生成するものがあります。詳細は各モジュールの仕様を参照ください。

3.11. メソッドリファレンス

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

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

```
image#delpalette():reduce
```

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

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

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

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

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

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

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

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

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

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

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

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

```
image#setalpha(alphaKey:number, colorKey?:color, alphaRest:number => 255):reduce
```

```
image#size()
```

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

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

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

3.12. iterator クラス

3.12.1. インスタンスの生成

- 多くの生成方法があります。「Gura 言語マニュアル」のイテレータの章を参照ください。

3.12.2. メソッドリファレンス

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リストのリファレンス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?}
```

```
iterator#isinfinite()
```

```
iterator#next()
```

3.13. list クラス

3.13.1. インスタンスの生成

- リスト宣言 `[…]` や `@{…}` でインスタンスを生成します。
- 関数 `list` でインスタンスを生成します。

3.13.2. メソッドリファレンス

```
list#add(elem+):reduce
```

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

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

```
list#append(elem+):reduce
```

```
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?}
```

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

条件に合致する要素の数を返します。条件 `criteria` には値または関数を指定します。

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

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

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

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

リストの要素を順に走査するイテレータを返します。ブロック引数を指定すると、要素ごとにブロックの内容を評価します。ブロック引数は `|value, idx:number|` となります。`value` が要素値、`idx` が走査インデクスです。

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

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

```
list#first()

list#flat()

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

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

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

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

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

list#isempty()

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

iterator#joinb()

list#last()

list#len() / iterator#len()

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

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

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

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

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

list#or() / iterator#or()

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

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

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

iterator#print(stream?:stream)

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

iterator#println(stream?:stream)

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

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

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

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

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

list#shift():[raise]

list#shuffle():reduce
```

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

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

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

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

`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?}`

3.14. matrix クラス

3.14.1. インスタンスの生成

- マトリクス宣言 `@@{…}` でインスタンスを生成します。
- 関数 `matrix` でインスタンスを生成します。

3.14.2. メソッドリファレンス

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

```
matrix#colsize()
```

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

```
matrix#eachcol()
```

```
matrix#eachrow()
```

```
matrix#inverse()
```

```
matrix#issquare()
```

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

```
matrix#rowsize()
```

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

```
matrix#setcol(col:number, value)
```

```
matrix#setrow(row:number, value)
```

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

```
matrix#tolist():[flat,transpose]
```

```
matrix#transpose()
```

3.15. palette クラス

3.15.1. インスタンスの生成

- 関数 palette でインスタンスを生成します。

3.15.2. メソッドリファレンス

```
palette#each() {block?}
```

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

```
palette#shrink():reduce
```

```
palette#updateby(image_or_palette):reduce:[shrink]
```

3.16. semaphore クラス

3.16.1. インスタンスの生成

- 関数 semaphore でインスタンスを生成します。

3.16.2. メソッドリファレンス

```
semaphore#release()
```

```
semaphore#session() {block}
```

```
semaphore#wait()
```

3.17. stream クラス

3.17.1. インスタンスの生成

- 関数 open でインスタンスを生成します。

3.17.2. メソッドリファレンス

```
stream#close()
```

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

```
stream#dosmode(dos_flag:boolean):reduce
```

```
stream#peek(len?:number)
```

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

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

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

```
stream#read(len?:number)
```

```
stream#readline():[chop]
```

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

```
stream#readtext()
```

```
stream#readto(stream:stream, bytesunit:number => 65536):map:reduce {block?}
```

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

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

```
stream#tell()
```

```
stream#write(buff:binary):reduce
```

```
stream#writefrom(stream:stream, bytesunit:number => 65536):map:reduce {block?}
```

3.18. string クラス

3.18.1. インスタンスの生成

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

3.18.2. メソッドリファレンス

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

```
string#capitalize()
```

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

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

```
string#encode(codec:codec)
```

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

```
string#escapehtml()
```

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

```
string#format(values*):map
```

```
string#isempty()
```

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

```
string#len()
```

```
string#lower()

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

string#print(stream?:stream):void

string#println(stream?:stream):void

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

string#right(len?:number):map

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

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

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

string#unescapehtml()

string#upper()
```

3.19. timedelta クラス

3.19.1. インスタンスの生成

- 関数 `time.delta` でインスタンスを生成します。
- `datetime` インスタンス同士を引き算した結果は `timedelta` インスタンスです。

3.19.2. メソッドリファレンス

4. sys モジュール

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。

4.1. 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。

`sys.listcodec()`

利用可能な文字コーデックの名前の一覧をリストで返します。

4.2. 変数

sys モジュールが定義・参照している変数は以下のとおりです。

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	Gura のバージョン番号
`build`	symbol	Gura をビルドした環境をシンボルで表す。`gcc: Linux 環境の gcc, `bcc: Windows 環境の Borland C、`msc: Windows 環境の Visual Studio
`executable`	string	Gura 実行ファイルのフルパス名
`datadir`	string	Gura のデータディレクトリのフルパス名
`libdir`	string	Gura の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

5. fs モジュール

ファイルシステムの操作をするモジュールです。このモジュールをインポートすることで、以下の関数の機能が拡張されます。

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

5.1. 関数

```
chdir(pathname:string):void
```

```
getcwd()
```

```
mkdir(pathname:string):map:void
```

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

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

5.2. fs.stat クラス

メンバ	データ型	内容
pathname	string	
basename	string	
dirname	string	
size	number	
uid	number	
gid	number	
atime	datetime	
mtime	datetime	
ctime	datetime	
isdir	boolean	
ischr	boolean	
isblk	boolean	
isreg	boolean	
isfifo	boolean	
islnk	boolean	
issock	boolean	

6. os モジュール

OS の操作をまとめたモジュールです。

6.1. 関数

```
exec(pathname:string, args*:string):map
```

```
getenv(name:string):map
```

```
putenv(name:string, value:string):void
```

7. path モジュール

パス操作をまとめたモジュールです。

7.1. 関数

```
absname(name:string):map:[http]
```

```
basename(pathname:string):map
```

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

```
dirname(pathname:string):map
```

```
exists(pathname:string):map
```

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

```
join(paths+:string):map:[uri]
```

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

```
split(pathname:string):map
```

```
splitext(pathname:string):map
```

```
stat(pathname:string):map
```

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

8. math モジュール

数学演算処理をまとめたモジュールです。

8.1. 関数

```
abs(num):map
```

```
acos(num):map
```

```
arg(num):map
```

```
asin(num):map
```

```
atan(num):map
```

```
atan2(num1, num2):map
```

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

```
ceil(num):map
```

```
conj(num):map
```

```
cos(num):map
```

```
cosh(num):map
```

```
covariance(a:iterator, b:iterator)
```

```
cross_product(a[], b[])
```

```
diff(expr:expr, var:symbol):map

dot_product(a[], b[])

exp(num):map

fft(seq[])

floor(num):map

imag(num):map

integral()

least_square(x:iterator, y:iterator, n:number => 1, var:symbol => x)

log(num):map

log10(num):map

norm(num):map

optimize(expr:expr):map

real(num):map

sin(num):map

sinh(num):map
```

```
sqrt(num):map
```

```
tan(num):map
```

```
tanh(num):map
```

9. string モジュール

文字列処理をまとめたモジュールです。」

9.1. 関数

10. time モジュール

時刻操作をまとめたモジュールです。

10.1. 関数

```
clock()
```

```
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
```

```
delta(days:number => 0, secs:number => 0, usecs:number => 0, msecs:number => 0,
    mins:number => 0, hours:number => 0, weeks:number => 0):map
```

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

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

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

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

```
sleep(secs:number)
```

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

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

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

11. hash モジュール

11.1. 関数

```
crc32(buff?:binary)
```

```
md5(buff?:binary)
```

```
sha1(buff?:binary)
```

12. ネットワーク – net.http モジュール

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。このモジュールをインポートすることで、以下の関数の機能が拡張されます。

- 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。

12.1. 関数

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

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

```
decquery(query:string)
```

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

```
splituri(uri:string)
```

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

13. イメージファイル – bmp モジュール

image クラスに image#read()および image#write()

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

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

14. イメージファイル – png モジュール

```
image#pngread(stream:stream):reduce
```

```
image#pngwrite(stream:stream):reduce
```

15. イメージファイル – jpeg モジュール

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

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

16. イメージファイル – msicon モジュール

```
image#msiconread(stream:stream):reduce
```

```
image#msiconwrite(stream:stream):reduce
```

17. イメージファイル – ppm モジュール

```
image#ppmread(stream:stream):reduce
```

```
image#ppmwrite(stream:stream):reduce
```

18. イメージファイル – gif モジュール

```
image#gifread(stream:stream):reduce
```

`image#gifwrite(stream:stream):reduce`

image#gif$gctl, image#gif$gdesc

gif.graphiccontrol

メンバ	データ型	内容
DisposalMethod	symbol	`none, `keep, `background, `previous
UserInputFlag	boolean	
TransparentColorFlag	boolean	
DelayTime	number	
TransparentColorIndex	number	

gif.imagedescriptor

メンバ	データ型	内容
ImageLeftPosition	number	
ImageTopPosition	number	
ImageWidth	number	
ImageHeight	number	
LocalColorTableFlag	boolean	
InterlaceFlag	boolean	
SortFlag	boolean	
SizeOfLocalColorTable	number	

メンバ	データ型	内容
hdr$Signature		
hdr$Version		
lsd$LogicalScreenWidth		
lsd$LogicalScreenHeight		
lsd$GlobalColorTableFlag		
lsd$ColorResolution		
lsd$SortFlag		
lsd$SizeOfGlobalColorTable		
lsd$BackgroundColorIndex		
lsd$BackgroundColor		
lsd$PixelAspectRatio		
cmmt$CommentData		

ptxt$TextGridLeftPosition		
ptxt$TextGridTopPosition		
ptxt$TextGridWidth		
ptxt$TextGridHeight		
ptxt$CharacterCellWidth		
ptxt$CharacterCellHeight		
ptxt$TextForegroundColorIndex		
ptxt$TextBackgroundColorIndex		
ptxt$PlainTextData		
ptxt$ApplicationIdentifier		
ptxt$AuthenticationCode		
ptxt$ApplicationData		

19. 二次元イメージ描画 – cairo モジュール

Cairo のオフィシャルサイトは http://cairographics.org/ です。

19.1. 命名規則

コンテキストは cairo.context クラスのインスタンスで表されます。コンテキストを操作する関数は、"cairo_" をとった名前のメソッドになります。例えば、cairo_set_source_rgb(cr, r, g, b) は cairo.context#set_source_rgb(r, g, b) となります。

パターンは cairo.pattern クラスのインスタンスで表されます。パターンを操作する関数は、"cairo_pattern_" をとった名前のメソッドになります。例えば、cairo_pattern_set_filter(pattern, filter) は pattern.set_filter(filter) となります。

グリフは cairo.glyph クラスのインスタンスで表されます。グリフを操作する関数は、"cairo_glyph_" とった名前のメソッドになります。例えば、cairo_glyph_

パスは cairo.path クラスのインスタンスで表されます。パスを操作する関数は、"cairo_path_" とった名前のメソッドになります。例えば、cairo_path_

サーフェースは cairo.surface クラスのインスタンスで表されます。サーフェースを操作する関数は、"cairo_surface_" とった名前のメソッドになります。例えば、cairo_surface_

デバイスは cairo.device クラスのインスタンスで表されます。デバイスを操作する関数は、"cairo_device_" をとった名前のメソッドになります。例えば、cairo_device_

フォントオプションは cairo.font_options クラスのインスタンスで表されます。フォントオプションを操作する関数は、"cairo_font_options_" をとった名前のメソッドになります。例えば、cairo_font_options_

矩形は cairo.rectangle クラスのインスタンスで表されます。矩形を操作する関数は、"cairo_rectangle_" をとった名前のメソッドになります。例えば、cairo_rectangle_

テキストエクステントは cairo.text_extents クラスのインスタンスで表されます。テキストエキステントを操作する関数は、"cairo_text_extents_"をとった名前のメソッドになります。例えば、cairo_text_extents_

フォントエキステントは cairo.font_extents クラスのインスタンスで表されます。フォントエキステントを操作する関数は、"cairo_font_extents_" をとった名前のメソッドになります。例えば、cairo_font_extents_

マクロ変数は、その変数が属するグループに共通してつけられるプレフィックスをとりのぞき、大文字をすべて小文字にした名前のシンボルで表されます。例えば、CAIRO_LINE_CAP_BUTT や CAIRO_LINE_CAP_ROUND は、それぞれシンボル `butt および `round で表されます。

19.2. 関数

```
emf_create(filename:string, width_in_points:number, height_in_points:number) {block?}
image_create(width:number, height:number, color?:color) {block?}
pattern_create_color(color:color, alpha?:number)
pattern_create_for_surface(surface:surface)
pattern_create_linear(x0:number, y0:number, x1:number, y1:number)
pattern_create_radial(cx0:number, cy0:number, radius0:number, cx1:number, cy1:number, radius1:number)
pattern_create_rgb(red:number, green:number, blue:number)
pattern_create_rgba(red:number, green:number, blue:number, alpha:number)
pdf_create(stream:stream, width_in_points:number, height_in_points:number) {block?}
ps_create(stream:stream, width_in_points:number, height_in_points:number) {block?}
svg_create(stream:stream, width_in_points:number, height_in_points:number) {block?}
```

19.3. cairo.surface クラス

```
surface#copy_page():reduce
surface#finish():reduce
surface#flush():reduce
surface#get_content()
surface#get_device()
surface#get_device_offset()
surface#get_fallback_resolution()
surface#get_font_options()
surface#get_mime_data()
surface#has_show_text_glyphs()
```

surface#mark_dirty():reduce

surface#mark_dirty_rectangle(x:number, y:number, width:number, height:number):reduce

surface#set_device_offset(x_offset:number, y_offset:number):reduce

surface#set_fallback_resolution(x_pixels_per_inch:number, y_pixels_per_inch:number):reduce

surface#set_mime_data():reduce

surface#show_page():reduce

19.4. cairo.pattern クラス

pattern#add_color_stop_rgb(offset:number, red:number, green:number, blue:number):reduce

pattern#add_color_stop_rgba(offset:number, red:number, green:number, blue:number, alpha:number):reduce

pattern#get_color_stop_count()

pattern#get_color_stop_rgba(index:number)

pattern#get_extend()

pattern#get_filter()

pattern#get_linear_points()

pattern#get_matrix()

pattern#get_radial_circles():reduce

pattern#get_rgba()

pattern#get_surface()

pattern#set_extend(extend:symbol):reduce

pattern#set_filter(filter:symbol):reduce

pattern#set_matrix(matrix:matrix):reduce

19.5. cairo.region クラス

region#contains_point(x:number, y:number)

region#contains_rectangle(rectangle:rectangle_int)

region#equal(region:region)

region#intersect(other:region)

region#intersect_rectangle(rectangle:rectangle_int)

region#is_empty()

region#subtract(other:region)

region#subtract_rectangle(rectangle:rectangle_int)

region#translate(dx:number, dy:number)

region#union(other:region)

region#union_rectangle(rectangle:rectangle_int)

region#xor(other:region)

region#xor_rectangle(rectangle:rectangle_int)

19.6. cairo.context クラス

context#arc(xc:number, yc:number, radius:number, angle1?:number, angle2?:number):map:reduce

context#arc_negative(xc:number, yc:number, radius:number, angle1?:number, angle2?:number):map:reduce

context#clip():reduce

context#clip_extents()

context#clip_preserve():reduce

context#close_path():reduce

context#copy_clip_rectangle_list()

context#copy_page():reduce

context#copy_path()

context#copy_path_flat()

context#curve_to(x1:number, y1:number, x2:number, y2:number, x3:number, y3:number):map:reduce

context#device_to_user(x:number, y:number)

context#device_to_user_distance(dx:number, dy:number)

context#fill():reduce

context#fill_extents():reduce

context#fill_preserve():reduce

```
context#font_extents()

context#get_antialias()

context#get_current_point()

context#get_dash()

context#get_fill_rule()

context#get_font_face()

context#get_font_matrix()

context#get_font_options()

context#get_group_target()

context#get_line_cap()

context#get_line_join()

context#get_line_width()

context#get_matrix()

context#get_miter_limit()

context#get_operator()

context#get_scaled_font()

context#get_source()

context#get_target()

context#get_tolerance()

context#glyph_extents()

context#has_current_point()

context#identity_matrix():reduce

context#in_clip(x:number, y:number)

context#in_fill(x:number, y:number)

context#in_stroke(x:number, y:number)

context#line_to(x:number, y:number):map:reduce
```

```
context#mask(pattern:pattern):reduce
context#mask_surface(surface:surface, surface_x:number, surface_y:number):reduce
context#move_to(x:number, y:number):map:reduce
context#new_path():reduce
context#new_sub_path():reduce
context#paint():reduce
context#paint_with_alpha(alpha:number):reduce
context#path_extents()
context#pop_group()
context#pop_group_to_source():reduce
context#push_group():reduce
context#push_group_with_content(content:symbol):reduce
context#rectangle(x:number, y:number, width:number, height:number):map:reduce
context#rel_curve_to(dx1:number, dy1:number, dx2:number, dy2:number, dx3:number, dy3:number):map:reduce
context#rel_line_to(dx:number, dy:number):map:reduce
context#rel_move_to(dx:number, dy:number):map:reduce
context#reset_clip():reduce
context#restore():reduce
context#rotate(angle:number):reduce
context#save():reduce {block?}
context#scale(sx:number, sy:number):reduce
context#select_font_face(family:string, slant:symbol, weight:symbol):reduce
context#set_antialias(antialias:symbol):reduce
context#set_dash(dashes[]:number, offset:number):reduce
context#set_fill_rule(fill_rule:symbol):reduce
context#set_font_face():reduce
```

```
context#set_font_matrix(matrix:matrix):reduce

context#set_font_options(options:font_options):reduce

context#set_font_size(size:number):reduce

context#set_line_cap(line_cap:symbol):reduce

context#set_line_join(line_join:symbol):reduce

context#set_line_width(width:number):reduce

context#set_matrix(matrix:matrix):reduce

context#set_miter_limit(limit:number):reduce

context#set_operator(op:symbol):reduce

context#set_scaled_font():reduce

context#set_source(source:pattern):reduce

context#set_source_color(color:color, alpha?:number):reduce

context#set_source_rgb(red:number, green:number, blue:number):reduce

context#set_source_rgba(red:number, green:number, blue:number, alpha:number):reduce

context#set_source_surface(surface:surface, x:number, y:number):reduce

context#set_tolerance(tolerance:number):reduce

context#show_glyphs(glyphs[]:glyph):reduce

context#show_page():reduce

context#show_text(text:string):reduce

context#show_text_glyphs():reduce

context#stroke():reduce

context#stroke_extents()

context#stroke_preserve():reduce

context#text_extents(text:string)

context#text_path(text:string):map:reduce

context#transform(matrix:matrix):reduce
```

```
context#translate(tx:number, ty:number):reduce
```

```
context#user_to_device(x:number, y:number)
```

```
context#user_to_device_distance(dx:number, dy:number)
```

20. canvas モジュール

21. フォント描画 – freetype モジュール

22. 三次元描画 – opengl および glu モジュール

OpenGL は…
オフィシャルサイトは http://www.opengl.org/ です。

22.1. 命名規則

23. 高速画面描画 – sdl モジュール

SDL は…
オフィシャルサイトは http://www.libsdl.org/ です。

23.1. 命名規則

23.2. sdl.Cursor クラス

```
Cursor#FreeCursor():void
```

23.3. sdl.Timer クラス

```
Timer#RemoveTimer()
```

23.4. sdl.PixelFormat クラス

```
PixelFormat#GetRGB(pixel:number)
PixelFormat#GetRGBA(pixel:number)
PixelFormat#MapRGB(r:number, g:number, b:number)
PixelFormat#MapRGBA(r:number, g:number, b:number, a:number)
```

23.5. sdl.PixelFormat クラス

```
Surface#ConvertSurface(fmt:PixelFormat, flag:number) {block?}
Surface#DisplayFormat() {block?}
Surface#DisplayFormatAlpha() {block?}
Surface#FillRect(rect:Rect, color:Color):map:void
```

```
Surface#Flip()
Surface#GetClipRect()
Surface#LockSurface()
Surface#PutSurface(src:Surface, x:number => 0, y:number => 0):map
Surface#SaveBMP(file:string):void
Surface#SetAlpha(flag:number, alpha:number)
Surface#SetClipRect(rect:Rect):map:void
Surface#SetColorKey(flag:number, key:number)
Surface#SetColors(colors[]:Color, firstcolor:number => 0)
Surface#SetPalette(flags:number, colors[]:Color, firstcolor:number => 0)
Surface#UnlockSurface():void
Surface#UpdateRect(x:number => 0, y:number => 0, w:number => 0, h:number =>
0):void
Surface#UpdateRects(rects[]:Rect):void
```

23.6. sdl.Overlay クラス

```
Overlay#DisplayYUVOverlay(dstrect:Rect)
Overlay#LockYUVOverlay()
Overlay#UnlockYUVOverlay():void
```

23.7. sdl.Joystick クラス

```
Joystick#JoystickClose():void
Joystick#JoystickGetAxis(axis:number)
Joystick#JoystickGetBall(ball:number)
Joystick#JoystickGetButton(button:number)
Joystick#JoystickGetHat(hat:number)
Joystick#JoystickIndex()
Joystick#JoystickNumAxes()
Joystick#JoystickNumBalls()
Joystick#JoystickNumButtons()
Joystick#JoystickNumHats()
```

23.8. sdl.AudioSpec クラス

```
AudioSpec#MixAudio(src:AudioSpec, volume:number)
```

23.9. sdl.AudioCVT クラス

```
AudioCVT#ConvertAudio()
```

23.10. sdl.CD クラス

```
CD#CDClose():void
CD#CDEject()
CD#CDPause()
CD#CDPlay(start:number, length:number)
CD#CDPlayTracks(start_track:number, start_frame:number, ntracks:number,
nframes:number)
CD#CDResume()
CD#CDStatus()
CD#CDStop()
CD#GetTrack(n:number):void
```

24. グラフィカルユーザインターフェース – tcl および tk モジュール

Tcl/Tk のオフィシャルサイトは http://www.tcl.tk/ です。

24.1. モジュール構成

モジュールは、Tcl インタープリタとのインターフェースを提供するモジュール `tcl.azd` と、そのモジュールを使って Tk の機能を実現するモジュール `tk.az` からなります。Tcl/Tk には多くの関数がありますが、`tcl.azd` や `tk.az` の中でそれらの関数を個別に実装しているわけではありません。ユーザが呼び出したメソッド名から動的にメソッドを生成して実行しています。

24.2. 命名規則

ウィジェットを生成する手続きは、親ウィジェットを表す変数のメソッドとして実装されます。
ウィジェットを生成する手続きは、最初の文字を大文字にした名前のメソッドになります。例えば、canvas は Canvas になります。
名前空間 ttk に属する手続きは、`"ttk$"` につづけて、最初の文字を大文字にした名前をつけたメソッドになります。例えば、`ttk::button` は `ttk$Button` になります。

24.3. イベントの扱い

Tcl/Tk は、イベントが発生すると関連付けられたプロシージャを評価します。この際、プロシージャ中にパーセント記号 `"%"` と一文字のアルファベットまたは記号が記述されていると、評価の前にその部分をイベントに付随するパラメータ値に置換します。例えば、マウスをクリックしたときに発生する ButtonPress イベントが発生すると、関連付けたプロシージャ中の `%x` や `%y` という部分をそれぞれマウスの x, y 座標数値に置換してから評価が行われます。置換文字列の一覧は http://www.tcl.tk/man/tcl8.5/TkCmd/bind.htm にまとめられています。

Gura はブロックでイベントハンドラを指定できますが、イベントのパラメータ値はブロック引数として受け取ります。このとき、ブロック引数の名前が、Tcl/Tk の置換文字列の名前に対応します。上の例を使えば、ブロック引数に x, y という名前の引数を指定すると、これらに x, y 座標値が入ります。一般の関数呼び出しの慣例では引

数の位置によって受け渡しする値を指定しますが、Tcl/Tk のイベントハンドラでは名前で受け取る値が決まることに注意してください。

また、Tcl/Tk が置換する結果は文字列になるので、数値など他の方で受け取りたい場合はブロック引数に明示的に型指定をしてください。

ブロック引数では、Tcl/Tk が定義する一文字の識別子のほかに、可読性を高めるため以下の別名も受け付けられるようにしています。

引数名	Tcl/Tk
serial	#
above	a
numbuttons	b
count	c
detail	d
user_data	d
focus	f
height	h
widget_num	i
keycode	k
mode	m
override_redirect	o

引数名	Tcl/Tk
place	p
state	s
time	t
width	w
x	x
y	y
char	A
border_width	B
delta	D
send_event	E
keysym	K
keysym_num	N

引数名	Tcl/Tk
root	R
subwidget	S
type	T
widget	W
x_root	X
y_root	Y
action	d
index	i
wouldbe	P
current	s
edited	S
validate	V

この対応づけは、tk.bindSubstDict 変数に辞書として登録されており、必要に応じて追加することができます。例えば、%h で置換されるイベントパラメータを hoge という引数名で受け取りたい場合は、以下のようなコードを追加します。

```
tk.bindSubstDict[`hoge] = "h"
```

24.4. モジュール関数

```
tk.bell(args*, opts%):map
tk.bind(args*, opts%):map
tk.bindtags(args*, opts%):map
tk.tkerror(args*, opts%):map
tk.update()
tk.winfo$atom(args*, opts%):map
tk.winfo$atomname(args*, opts%):map
tk.winfo$containing(args*, opts%):map
tk.winfo$interps(args*, opts%):map
tk.winfo$pathname(args*, opts%):map
tk.tk$FocusFollowsMouse(args*, opts%):map
tk.tk$SetPalette(args*, opts%):map
```

```
tk.tk$bisque(args*, opts%):map
tk.tkwait$variable(args*, opts%):map
tk.tkwait$visibility(args*, opts%):map
tk.tkwait$window(args*, opts%):map
tk.mainloop()
tk.tclDump(flag:boolean => true)
```

24.5. image クラスへの追加メソッド

tk モジュールをインポートすると、image クラスに以下のメソッドが追加されます。

```
image#to_tk()
```

24.6. tk.Widget クラス

```
Widget#wm$aspect(minNumber?, minDenom?, maxNumber?, maxDenom?):map
```

Tcl コマンド: `wm aspect window ?minNumber mindenom maxNumber maxDenom`

```
Widget#wm$attributes(opts%):map
```

Tcl コマンド: `wm attributes window`

```
Widget#wm$client(name?):map
```

Tcl コマンド: `wm client window ?name?`

```
Widget#wm$colormapwindows(windowList?):map
```

Tcl コマンド:

```
Widget#wm$command(value?):map
```

Tcl コマンド:

```
Widget#wm$deiconify():map
```

Tcl コマンド:

```
Widget#wm$focusmodel(args*, opts%):map
```

Tcl コマンド:

```
Widget#wm$forget():map
```

Tcl コマンド:

```
Widget#wm$frame():map
```

Tcl コマンド:

```
Widget#wm$geometry(newGeometry):map
```

Tcl コマンド:

```
Widget#wm$grid(baseWidth?, baseHeight?, widthInc?, heightInc?):map
```

Tcl コマンド:

Widget#wm$group(args*, opts%):map

Tcl コマンド:

Widget#wm$iconbitmap(args*, opts%):map

Tcl コマンド:

Widget#wm$iconify(args*, opts%):map

Tcl コマンド:

Widget#wm$iconmask(args*, opts%):map

Tcl コマンド:

Widget#wm$iconname(args*, opts%):map

Tcl コマンド:

Widget#wm$iconphoto(args*, opts%):map

Tcl コマンド:

Widget#wm$iconposition(args*, opts%):map

Tcl コマンド:

Widget#wm$iconwindow(args*, opts%):map

Tcl コマンド:

Widget#wm$manage(args*, opts%):map

Tcl コマンド:

Widget#wm$maxsize(args*, opts%):map

Tcl コマンド:

Widget#wm$minsize(args*, opts%):map

Tcl コマンド:

Widget#wm$overrideredirect(args*, opts%):map

Tcl コマンド:

Widget#wm$positionfrom(args*, opts%):map

Tcl コマンド:

Widget#wm$protocol(args*, opts%):map

Tcl コマンド:

Widget#wm$resizable(args*, opts%):map

Tcl コマンド:

```
Widget#wm$sizefrom(args*, opts%):map
```

Tclコマンド:

```
Widget#wm$stackorder(args*, opts%):map
```

Tclコマンド:

```
Widget#wm$state(args*, opts%):map
```

Tclコマンド:

```
Widget#wm$title(args*, opts%):map
```

Tclコマンド:

```
Widget#wm$transient(args*, opts%):map
```

Tclコマンド:

```
Widget#wm$withdraw():map
```

Tclコマンド:

```
Widget#winfo$(args*, opts%):map
```

Tclコマンド:

```
Widget#grid$(args*, opts%):map
```

Tclコマンド:

```
Widget#place$(args*, opts%):map
```

Tclコマンド:

```
Widget#tk$bisque(args*, opts%):map
```

Tclコマンド:

```
Widget#tk$chooseColor(args*, opts%):map
```

Tclコマンド:

```
Widget#tk$chooseDirectory(args*, opts%):map
```

Tclコマンド:

```
Widget#tk$dialog(args*, opts%):map
```

Tclコマンド:

```
Widget#tk$focusFollowsMouse(args*, opts%):map
```

Tclコマンド:

```
Widget#tk$focusNext(args*, opts%):map
```

Tclコマンド:

```
Widget#tk$focusPrev(args*, opts%):map
```

Tcl コマンド:

Widget#tk$getOpenFile(args*, opts%):map

Tcl コマンド:

Widget#tk$getSaveFile(args*, opts%):map

Tcl コマンド:

Widget#tk$menuSetFocus(args*, opts%):map

Tcl コマンド:

Widget#tk$messageBox(args*, opts%):map

Tcl コマンド:

Widget#tk$optionMenu(args*, opts%):map

Tcl コマンド:

Widget#tk$popup(args*, opts%):map

Tcl コマンド:

Widget#tk$setPalette(args*, opts%):map

Tcl コマンド:

Widget#tk$textCopy(args*, opts%):map

Tcl コマンド:

Widget#tk$textCut(args*, opts%):map

Tcl コマンド:

Widget#tk$textPaste(args*, opts%):map

Tcl コマンド:

24.6.1. ウィジェット生成メソッド

例えば、button ウィジェットを生成するメソッドの一般式は以下のようになります。

```
tk.Widget#Button(options%)
```

このメソッドは、Tcl コマンドとして"button pathName %options%" を実行します。pathName は新しく生成する button ウィジェットのパス名、options は tk.Widget#Button メソッドに辞書形式で渡した引数の内容が渡されます。

ウィジェット生成メソッドの戻り値は、新しく生成したウィジェットのパス名を持った tk.Widget インスタンスです。menu ウィジェットや text ウィジェットのように、ウィジェット特有のメソッドを持つものについては、tk.Widget ク

ラスから派生したクラスのインスタンスを返します。

ウィジェット生成メソッドと、生成する Tk ウィジェットの対応表を以下にまとめます。

メソッド	生成する Tk ウィジェット	メソッドの戻り値
tk.Widget#Button	button	tk.Widget
tk.Widget#Canvas	canvas	tk.Widget
tk.Widget#Checkbutton	checkbutton	tk.Widget
tk.Widget#Entry	entry	tk.Widget
tk.Widget#Frame	frame	tk.Widget
tk.Widget#Label	label	tk.Widget
tk.Widget#Labelframe	labelframe	tk.Widget
tk.Widget#Listbox	listbox	tk.Widget
tk.Widget#Menu	menu	tk.Menu
tk.Widget#Menubutton	menubutton	tk.Widget
tk.Widget#Message	message	tk.Widget
tk.Widget#Panedwindow	panedwindow	tk.Widget
tk.Widget#Radiobutton	radiobutton	tk.Widget
tk.Widget#Scrollbar	scrollbar	tk.Widget
tk.Widget#Spinbox	spinbox	tk.Widget
tk.Widget#Text	text	tk.Text
tk.Widget#Toplevel	toplevel	tk.Widget
tk.Widget#ttk$Button	ttk::button	tk.Widget
tk.Widget#ttk$Checkbutton	ttk::checkbutton	tk.Widget
tk.Widget#ttk$Combobox	ttk::combobox	tk.Widget
tk.Widget#ttk$Entry	ttk::entry	tk.Widget
tk.Widget#ttk$Frame	ttk::frame	tk.Widget
tk.Widget#ttk$Intro	ttk::intro	tk.Widget
tk.Widget#ttk$Label	ttk::label	tk.Widget
tk.Widget#ttk$Labelframe	ttk::labelframe	tk.Widget
tk.Widget#ttk$Menubutton	ttk::menubutton	tk.Widget
tk.Widget#ttk$Notebook	ttk::notebook	tk.Notebook
tk.Widget#ttk$Panedwindow	ttk::panedwindow	tk.Widget
tk.Widget#ttk$Progressbar	ttk::progressbar	tk.Widget
tk.Widget#ttk$Radiobutton	ttk::radiobutton	tk.Widget
tk.Widget#ttk$Scale	ttk::scale	tk.Widget
tk.Widget#ttk$Scrollbar	ttk::scrollbar	tk.Widget
tk.Widget#ttk$Separator	ttk::separator	tk.Widget
tk.Widget#ttk$Sizegrip	ttk::sizegrip	tk.Widget

tk.Widget#ttk$Spinbox	ttk::spinbox	tk.Widget
tk.Widget#ttk$Style	ttk::style	tk.Widget
tk.Widget#ttk$Treeview	ttk::treeview	tk.Treeview
tk.Widget#ttk$Widget	ttk::widget	tk.Widget
tk.Widget#ttk$Image	ttk::image	tk.Widget
tk.Widget#ttk$Vsapi	ttk::vsapi	tk.Widget

24.7. tk.Menu クラス

```
Menu#Separator(opts%)
Menu#Cascade(opts%) {block?}
Menu#Command(opts%) {block?}
Menu#Checkbutton(opts%) {block?}
Menu#Radiobutton(opts%) {block?}
```

24.8. tk.Canvas クラス

```
Canvas#Tag() {block?}
```

24.9. tk.CanvasItem クラス

24.10. tk.CanvasTag クラス

24.11. tk.Text クラス

24.12. tk.TextTag クラス

24.13. tk.Notebook クラス

24.14. tk.Treeview クラス

24.15. tk.TreeviewItem クラス

24.16. tk.TreeviewTag クラス

24.17. tk.FontT クラス

24.18. tk.Image クラス

24.19. 注意事項

Tk イメージオブジェクトは、明示的に destroy() を実行しないとメモリから削除されません。

25. データベース – sqlite3 モジュール

25.1. 関数

```
open(filename:string) {block?}
```

25.2. sqlite3 クラス

```
sqlite3.sqlite3#close()
```

```
sqlite3.sqlite3#exec(sql:string):map
```

```
sqlite3.sqlite3#getcolnames(sql:string):map {block?}
```

```
sqlite3.sqlite3#query(sql:string):map {block?}
```

```
sqlite3.sqlite3#setprop!(symbol:symbol, value):map
```

```
sqlite3.sqlite3#transaction() {block}
```

26. オーディオ

26.1. wav

27. アーカイブファイル – gzip モジュール

```
gzip.open(name:string, mode?:string, encoding:string => "utf-8")
```

```
gzip.decomp(stream:stream, winbits?:number)
```

```
gzip.comp(stream:stream, winbits?:number)
```

```
stream#gzipdecomp(winbits?:number)
```

```
stream#gzipcomp(winbits?:number)
```

28. アーカイブファイル – bzip2 モジュール

```
bzip2.open(name:string, mode?:string, encoding:string => "utf-8")
```

```
bzip2.decomp(stream:stream)
```

```
bzip2.comp(stream:stream)
```

```
stream#bzip2decomp()
```

```
stream#bzip2comp()
```

29. アーカイブファイル – zipfile モジュール

ZIP アーカイブの操作をするモジュールです。このモジュールをインポートすることで、以下の関数の機能が拡張されます。

- 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。

29.1. 関数

```
zipfile.open(pathname:string) {block?}
```

```
zipfile.create(pathname:string) {block?}
```

29.2. zipfile.zipfile クラス

```
zipfile.zipfile#add(stream:stream, filename?:string):map:reduce
```

```
zipfile.zipfile#each() {block?}
```

```
zipfile.zipfile#close():reduce
```

29.3. zipfile.stat クラス

メンバ	データ型	内容
filename	string	ファイル名
comment	string	コメント

mtime	datetime	最終更新日時
crc32	number	CRC32 チェックサム
compression_method	number	圧縮形式
size	number	圧縮前のサイズ
compressed_size	number	圧縮後のサイズ
attributes	number	アトリビュート

30. アーカイブファイル – tar モジュール

TAR アーカイブの操作をするモジュールです。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。このモジュールをインポートすることで、以下の関数の機能が拡張されます。

- 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。

30.1. 関数

```
tar.open(pathname:string) {block?}
```

```
tar.create(pathname:string) {block?}
```

30.2. tar.tar クラス

```
tar.tar#add(stream:stream, filename?:string):map:reduce
```

```
tar.tar#each() {block?}
```

```
tar.tar#close():reduce
```

30.3. tar.stat クラス

メンバ	データ型	説明
name	string	ファイル名
linkname	string	
uname	string	
gname	string	

mode	number	
uid	number	
gid	number	
size	number	
mtime	datetime	
atime	datetime	
ctime	datetime	
chksum	number	
typeflag	number	
devmajor	number	
devminor	number	

31. テキスト処理 – re モジュール

正規表現のエンジンとして鬼車（http://www.geocities.jp/kosako3/oniguruma/）を使用しています。正規表現パターンの詳細は、上記のホームページをご覧ください。

正規表現処理では、一つのファンクションで 3 つの呼び出し形式があります。状況に応じて適切な表記方法を選んでください。

関数	re.pattern メソッド	string メソッド
re.match	re.pattern#match	string#match
re.scan	re.pattern#scan	string#scan
re.split	re.pattern#split	string#splitreg
re.sub	re.pattern#sub	string#sub

31.1. 関数

```
re.match(pattern:pattern, str:string, pos:number => 0, endpos?:number):map
```

re.match インスタンスを返します。

```
re.pattern(pattern:string):map:[icase,multiline]
```

re.pattern インスタンスを返します。

```
re.scan(pattern:pattern, str:string, pos:number => 0, endpos?:number):map {block?}
```

```
re.split(pattern:pattern, str:string, count?:number):map {block?}
```

```
re.sub(pattern:pattern, replace, str:string, count?:number):map
```

31.2. re.match クラス

```
re.match#end(index):map
```

```
re.match#group(index):map
```

```
re.match#groups()
```

```
re.match#start(index):map
```

31.3. re.pattern クラス

```
re.pattern#match(str:string, pos:number => 0, endpos?:number):map
```

```
re.pattern#scan(str:string, pos:number => 0, endpos?:number):map {block?}
```

```
re.pattern#split(str:string, count?:number):map {block?}
```

```
re.pattern#sub(replace, str:string, count?:number):map
```

31.4. string クラスへの追加メソッド

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:regex, pos:number => 0, endpos?:number):map
```

```
string#sub(pattern:regex, replace, count?:number):map
```

```
string#splitreg(pattern:regex, count?:number):map {block?}
```

```
string#scan(pattern:regex, pos:number => 0, endpos?:number):map {block?}
```

32. テキスト処理 – csv モジュール

CSV ファイルの読み書きを行います。

32.1. 関数

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

```
csv.read(stream:stream) {block?}
```

```
csv.write(stream:stream, elem[])
```

```
csv.writes(stream:stream, iter:iterator)
```

32.2. stream クラスへの追加メソッド

```
stream#csvread() {block?}
```

33. テキスト処理 – xml モジュール

XML ファイルの読み書きを行います。

33.1. 関数

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

```
xml.parser() {block}
```

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

33.2. stream クラスへの追加メソッド

```
stream#xmlread()
```

34. テキスト処理 – yaml モジュール

YAML ファイルの読み書きを行います。YAML ファイルのフォーマットについてはオフィシャルサイト http://www.yaml.org/ を参照ください。

34.1. 関数

```
yaml.compose(obj)
```

```
yaml.parse(str:string)
```

```
yaml.read(stream:stream)
```

```
yaml.write(stream:stream, obj):reduce
```

34.2. ストリームクラスへの追加メソッド

```
stream#yamlread()
```

```
stream#yamlwrite()
```

35. プラットフォーム – win32 モジュール

```
win32.olecreate(progid:string):map:[import_const]
```

```
win32.oleconnect(progid:string):map:[import_const]
```

35.1. win32.regkey クラス

```
win32.HKEY_CLASSES_ROOT
win32.HKEY_CURRENT_CONFIG
win32.HKEY_CURRENT_USER
win32.HKEY_LOCAL_MACHINE
win32.HKEY_USERS
win32.HKEY_PERFORMANCE_DATA
win32.HKEY_DYN_DATA
```

```
regkey#createkey(subkey:string, option?:symbol, samDesired[]?:symbol:nomap):map {block?}
```

```
regkey#deletekey(subkey:string):map:void
```

```
regkey#deletevalue(valueName:string):map:void
```

```
regkey#enumkey(samDesired[]?:symbol):[openkey] {block?}
```

```
regkey#enumvalue()
```

```
regkey#openkey(subkey:string, samDesired[]?:symbol:nomap):map {block?}
```

```
regkey#queryvalue(valueName?:string):map
```

```
regkey#setvalue(valueName:string, data:nomap):map
```

36. プラットフォーム – uuid モジュール

```
uuid.generate():[upper]
```

37. 開発ツール

37.1. lets_module

コマンドラインから使用します。「Gura 開発者向けマニュアル」を参照ください。

37.2. module_builder