Gura ライブラリリファレンス

Updated: April 18, 2013

copyright © 2011- Yutaka Saito ([ypsitau@nifty.com](mailto:ypsitau@nifty.com))

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

目次

1. このリファレンスについて 11

2. 組込み関数 12

2.1. テキスト表示 12

2.2. 制御構文 13

2.2.1. 関数内のフロー制御 13

2.2.2. 繰り返し 13

2.2.3. 繰り返し中のフロー制御 14

2.2.4. 条件分岐 14

2.2.5. 例外処理 14

2.2.6. switch文 15

2.3. データ変換 15

2.4. クラス操作 16

2.5. 変数スコープ操作 16

2.6. イテレータ生成 17

2.7. フォーマット変換 17

2.8. モジュール 18

2.9. データ型チェック 18

2.10. 演算・統計 19

2.11. スクリプト評価 19

2.12. 乱数 19

2.13. その他 20

3. 定義済み変数 21

4. 組込みクラス 22

4.1. objectクラス 22

4.1.1. 概要 22

4.1.2. インスタンスの生成 22

4.2. binaryクラス 22

4.2.1. 概要 22

4.2.2. インスタンスの生成 22

4.2.3. クラスメソッド 22

4.2.4. インスタンスメソッド 23

4.3. codecクラス 25

4.3.1. 概要 25

4.3.2. インスタンスの生成 25

4.3.3. クラスメソッド 25

4.3.4. インスタンスメソッド 25

4.4. colorクラス 26

4.4.1. 概要 26

4.4.2. インスタンスの生成 26

4.4.3. Web標準カラー名 26

4.4.4. クラスプロパティ 26

4.4.5. インスタンスプロパティ 26

4.4.6. インスタンスメソッド 27

4.5. dictクラス 27

4.5.1. インスタンスの生成 27

4.5.2. インスタンスメソッド 27

4.6. environmentクラス 29

4.6.1. インスタンスの生成 29

4.6.2. インスタンスメソッド 29

4.7. errorクラス 29

4.7.1. インスタンスの生成 29

4.7.2. インスタンスメソッド 30

4.8. exprクラス 30

4.8.1. インスタンスの生成 30

4.8.2. Expr要素と判定メソッド 30

4.8.3. 要素を参照するインスタンスメソッド 31

4.8.4. その他のインスタンスメソッド 31

4.8.5. 式を構成する要素 32

4.9. functionクラス 34

4.9.1. インスタンスの生成 34

4.9.2. インスタンスプロパティ 34

4.9.3. インスタンスメソッド 34

4.10. imageクラス 35

4.10.1. インスタンスの生成 35

4.11. インスタンスメソッド 35

4.12. iteratorクラス 37

4.12.1. インスタンスの生成 37

4.12.2. インスタンスメソッド 38

4.13. listクラス 38

4.13.1. インスタンスの生成 38

4.13.2. インスタンスメソッド 39

4.14. matrixクラス 45

4.14.1. インスタンスの生成 45

4.14.2. インデクスによる要素操作 45

4.14.3. インスタンスメソッド 46

4.15. paletteクラス 47

4.15.1. インスタンスの生成 47

4.15.2. インスタンスメソッド 47

4.16. semaphoreクラス 48

4.16.1. インスタンスの生成 48

4.16.2. インスタンスメソッド 48

4.17. streamクラス 48

4.17.1. インスタンスの生成 48

4.17.2. ストリーム操作を行うグローバル関数 48

4.17.3. インスタンスメソッド 49

4.17.4. インスタンスプロパティ 51

4.18. stringクラス 51

4.18.1. インスタンスの生成 51

4.18.2. インスタンスメソッド 51

5. sysモジュール 55

5.1. 概要 55

5.2. モジュール関数 55

5.3. モジュール変数 55

6. fsモジュール 56

6.1. 概要 56

6.2. ストリームのオープン 56

6.3. パスのサーチ 56

6.4. モジュール関数 56

6.5. fs.statクラス 57

6.5.1. インスタンスプロパティ 57

7. osモジュール 58

7.1. 概要 58

7.2. モジュール変数 58

7.3. モジュール関数 58

8. pathモジュール 60

8.1. 概要 60

8.2. モジュール関数 60

9. mathモジュール 63

9.1. 概要 63

9.2. モジュール関数 63

10. timeモジュール 66

10.1. 概要 66

10.2. モジュール関数 66

10.3. モジュール変数 66

10.4. datetimeクラス 66

10.4.1. 概要 66

10.4.2. インスタンスの生成 66

10.4.3. インスタンスメソッド 67

10.4.4. インスタンスプロパティ 68

10.5. timedeltaクラス 68

10.5.1. 概要 68

10.5.2. インスタンスの生成 68

10.5.3. インスタンスプロパティ 68

11. hashモジュール 69

11.1. 概要 69

11.2. モジュール関数 69

12. net.httpモジュール 70

12.1. 概要 70

12.2. パス名の拡張 70

12.3. モジュール変数 70

12.4. モジュール関数 70

12.5. net.http.serverクラス 71

12.5.1. インスタンスの生成 71

12.5.2. インスタンスプロパティ 71

12.5.3. インスタンスメソッド 71

12.5.4. サンプルプログラム 71

12.6. net.http.clientクラス 71

12.6.1. インスタンスの生成 71

12.6.2. インスタンスメソッド 72

12.6.3. リクエスト発行インスタンスメソッド 72

12.6.4. サンプルプログラム 73

12.7. net.http.statクラス 73

12.7.1. 概要 73

12.7.2. メッセージヘッダのフィールド定義 73

12.7.3. インスタンスプロパティ 73

12.7.4. インスタンスメソッド 73

12.8. net.http.requestクラス 73

12.8.1. 概要 73

12.8.2. メッセージヘッダのフィールド定義 74

12.8.3. インスタンスプロパティ 74

12.8.4. インスタンスメソッド 74

12.9. net.http.sessionクラス 75

12.9.1. 概要 75

12.9.2. インスタンスプロパティ 75

12.10. net.http.responseクラス 76

12.10.1. 概要 76

12.10.2. メッセージヘッダのフィールド定義 76

12.10.3. インスタンスプロパティ 76

12.10.4. インスタンスメソッド 76

13. bmpモジュール 78

13.1. 概要 78

13.2. ストリーム処理 78

13.3. imageクラスの拡張 78

13.3.1. インスタンスメソッド 78

14. gifモジュール 79

14.1. 概要 79

14.2. ストリームの読み書き 79

14.3. gif.contentクラス 79

14.3.1. 概要 79

14.3.2. GIF Data Streamの構造 79

14.3.3. 制限事項 80

14.3.4. インスタンスの生成 80

14.3.5. インスタンスメソッド 80

14.3.6. インスタンスプロパティ 81

14.3.7. インスタンスプロパティの詳細 81

14.4. imageクラスの拡張 83

14.4.1. インスタンスメソッド 83

14.4.2. インスタンスプロパティ 83

14.4.3. インスタンスプロパティの詳細 83

14.4.4. パレットの扱い 84

15. jpegモジュール 86

15.1. 概要 86

15.2. ストリームの読み書き 86

15.3. imageクラスの拡張 86

15.3.1. インスタンスメソッド 86

16. msicoモジュール 87

16.1. 概要 87

16.2. ストリームの読み書き 87

16.3. msico.contentクラス 87

16.3.1. 概要 87

16.3.2. インスタンスの生成 87

16.3.3. インスタンスメソッド 87

16.4. imageクラスの拡張 88

16.4.1. インスタンスメソッド 88

17. pngモジュール 89

17.1. 概要 89

17.2. ストリームの読み書き 89

17.3. imageクラスの拡張 89

17.3.1. インスタンスメソッド 89

18. ppmモジュール 90

18.1. 概要 90

18.2. ストリームの読み書き 90

18.3. imageクラスの拡張 90

18.3.1. インスタンスメソッド 90

19. xpmモジュール 91

19.1. 概要 91

19.2. ストリームの書きこみ 91

19.3. imageクラスの拡張 91

19.3.1. インスタンスメソッド 91

20. freetype モジュール 92

20.1. 概要 92

20.2. 関数 92

20.3. freetype.faceクラス 92

20.3.1. インスタンスメソッド 92

20.3.2. インスタンスプロパティ 93

20.4. imageクラスの拡張 94

20.4.1. インスタンスメソッド 94

21. sqlite3モジュール 95

21.1. 概要 95

21.2. データオブジェクトの対応 95

21.3. sqlite3.dbクラス 95

21.3.1. インスタンスの生成 95

21.3.2. インスタンスメソッド 95

22. gzip モジュール 96

22.1. 概要 96

22.2. モジュール変数 96

22.3. モジュール関数 96

22.4. streamクラスの拡張 96

22.4.1. インスタンスメソッド 96

23. bzip2 モジュール 97

23.1. 概要 97

23.2. モジュール関数 97

23.3. streamクラスの拡張 97

23.3.1. インスタンスメソッド 97

24. zip モジュール 98

24.1. 概要 98

24.2. パス名の拡張 98

24.3. zip.readerクラス 98

24.3.1. インスタンスの生成 98

24.3.2. インスタンスメソッド 98

24.4. zip.writerクラス 98

24.4.1. インスタンスの生成 98

24.4.2. インスタンスメソッド 99

24.5. zip.statクラス 99

24.5.1. インスタンスプロパティ 99

25. tar モジュール 100

25.1. 概要 100

25.2. パス名の拡張 100

25.3. モジュール変数 100

25.4. tar.readerクラス 101

25.4.1. インスタンスの生成 101

25.4.2. インスタンスメソッド 101

25.5. tar.writerクラス 101

25.5.1. インスタンスの生成 101

25.5.2. インスタンスメソッド 102

25.6. tar.statクラス 102

25.6.1. インスタンスプロパティ 102

26. reモジュール 103

26.1. 概要 103

26.2. 正規表現パターン記述について 103

26.3. モジュール関数 103

26.4. re.matchクラス 104

26.4.1. インスタンスの生成 104

26.4.2. マッチパターンの取得 104

26.4.3. インスタンスメソッド 105

26.5. re.patternクラス 105

26.5.1. インスタンスの生成 105

26.5.2. インスタンスメソッド 105

26.6. stringクラスの拡張 106

26.6.1. インスタンスメソッド 106

27. csvモジュール 107

27.1. 概要 107

27.2. モジュール関数 107

27.3. csv.writerクラス 107

27.3.1. インスタンスプロパティ 107

27.3.2. インスタンスメソッド 107

27.4. streamクラスの拡張 107

27.4.1. インスタンスメソッド 107

28. xmlモジュール 108

28.1. 概要 108

28.2. モジュール関数 108

28.3. xml.parserクラス 108

28.3.1. インスタンスの生成 108

28.3.2. オーバーライドメソッド 108

28.3.3. インスタンスプロパティ 109

28.3.4. インスタンスメソッド 109

28.4. xml.elementクラス 109

28.4.1. インスタンスの生成 109

28.5. streamクラスの拡張 110

28.5.1. インスタンスメソッド 110

29. yamlモジュール 111

29.1. 概要 111

29.2. データオブジェクトの対応 111

29.3. モジュール関数 111

29.4. streamクラスの拡張 111

29.4.1. インスタンスメソッド 111

30. uuid モジュール 112

30.1. 概要 112

30.2. モジュール関数 112

31. mswinモジュール 113

31.1. 概要 113

31.2. mswin.oleクラス 113

31.2.1. インスタンスの生成 113

31.3. mswin.regkeyクラス 113

31.3.1. 概要 113

31.3.2. 定義済みインスタンス 113

31.3.3. インスタンスメソッド 113

31.4. COMについて 115

31.4.1. COMサーバへの接続 115

31.4.2. プロパティの取得 115

31.4.3. プロパティの設定 116

31.4.4. メソッドの実行 116

31.4.5. イテレータの生成 116

32. lets\_moduleモジュール 117

32.1. 概要 117

33. modbuildモジュール 118

33.1. 概要 118

33.2. modbuild.Builderクラス 118

33.3. インスタンスプロパティ 118

33.4. インスタンスメソッド 118

34. gurcbuildモジュール 119

34.1. 概要 119

34.2. モジュール関数 119

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

組込み関数

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。

制御構文

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。

繰り返し

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

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

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

*symbol* in *iterator*

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

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

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

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

|  |  |
| --- | --- |
| スクリプト | 実行結果 |
| cross (x in 0..1, y in 0..2) {  printf('[%d,%d]', x, y)  } | [0,0][0,1][0,2][1,0][1,1][1,2] |

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。

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')  } |

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except関数を実行します。

except(errors\*:error):leader:trailer {block}  
tryまたはexcept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  } except(ValueError) {|e|  println('ValueError captured: ', e.text)  } except(IOError) {|e|  println('IOError captured: ', e.text)  } except {|e|  println('other error captured: ', e.text)  } |

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。

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。

クラス操作

class(superclass?:function) {block?}  
クラスを生成します。詳細は「Gura言語マニュアル」を参照ください。

struct(`args+):[loose] {block?}  
構造体のコンストラクタ関数を生成します。詳細は「Gura言語マニュアル」を参照ください。

classref(type:expr):map {block?}  
指定の型のクラスへの参照を返します。

super(obj):map {block?}  
スーパークラスのメソッドや変数を参照するオブジェクトを返します。

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。

イテレータ生成

fill(n:number, value?) {block?}  
引数nで指定した数だけ同じ値valueを返すイテレータを生成します。引数valueを省略するとnil値になります。  
blockをつけると、要素ごとにその内容を評価します。ブロックパラメータの形式は |value, idx:number| で、valueに関数に渡したvalueの値、idxに0から始まるインデクス番号が入ります。

interval(a:number, b:number, samples:number):map:[open,open\_l,open\_r] {block?}  
条件a ≤ x ≤ 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 ≤ b  
 :open\_r a ≤ x< b  
 :open a < x < b　(アトリビュートに :open\_l:open\_r と指定したのと同じです)  
blockをつけると、要素ごとにその内容を評価します。ブロックパラメータの形式は |num:number, idx:number| で、numに生成した数値、idxに0から始まるインデクス番号が入ります。

range(num:number, num\_end?:number, step?:number):map {block?}  
引数numのみを指定すると、0からnum-1までの整数を生成するイテレータを返します。引数numとnum\_endを指定すると、numからnum\_end-1 までの整数を生成するイテレータを返します。引数stepで数値の間隔を指定します。省略すると間隔が1になります。  
blockをつけると、要素ごとにその内容を評価します。ブロックパラメータの形式は |num:number, idx:number| で、numに生成した数値、idxに0から始まるインデクス番号が入ります。

iterator(value+) {block?}  
引数に指定したデータからイテレータを生成し、それを結合したイテレータを返します。  
blockをつけると、要素ごとにその内容を評価します。ブロックパラメータの形式は |value, idx:number| で、valueに要素の値、idxに0から始まるインデクス番号が入ります。

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。

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。

データ型チェック

istype(value, type:expr):map

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

|  |  |
| --- | --- |
| 関数 | 等価な呼び出し |
| isbinary(value) | istype(value, `binary) |
| isboolean(value) | istype(value, `boolean) |
| isclass(value) | istype(value, `class) |
| iscomplex(value) | istype(value, `complex) |
| isdatetime(value) | istype(value, `datetime) |
| isdict(value) | istype(value, `dict) |
| isenvironment(value) | istype(value, `environment) |
| iserror(value) | istype(value, `error) |
| isexpr(value) | istype(value, `expr) |
| isfunction(value) | istype(value, `function) |
| isiterator(value) | istype(value, `iterator) |
| islist(value) | istype(value, `list) |
| ismatrix(value) | istype(value, `matrix) |
| ismodule(value) | istype(value, `module) |
| isnumber(value) | istype(value, `number) |
| issemaphore(value) | istype(value, `semaphore) |
| isstring(value) | istype(value, `string) |
| issymbol(value) | istype(value, `symbol) |
| istimedelta(value) | istype(value, `timedelta) |
| isuri(value) | istype(value, `uri) |

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

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

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

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。

スクリプト評価

eval(expr:expr):map  
引数exprの内容を現在の環境で評価し、その結果を返します。

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(num?:number, range?: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。

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。

定義済み変数

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

組込みクラス

objectクラス

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。

インスタンスの生成

object()  
object型インスタンスを生成します。

complexクラス

### 概要

複素数を扱うクラスです。

インスタンスメソッド

complex#abs()  
複素数の絶対値を返します。

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

complex#imag()  
複素数の虚数成分を返します。

complex#norm()  
複素数のノルム値を返します。

complex#real()  
複素数の実数成分を返します。

fractionクラス

### 概要

分数を扱うクラスです。

インスタンスメソッド

fraction#denominator()  
分母を返します。

fraction#numerator()  
分子を返します。

fraction#reduce()  
分子と分母を通分した結果を返します。

binaryクラス

### 概要

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

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

インスタンスの生成

コンストラクタ関数binaryを使ってインスタンスを生成します。

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

クラスメソッド

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

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

|  |  |
| --- | --- |
| 指定子 | 説明 |
| x | データを埋め込まず、アドレスを指定のバイト数分だけ進めます。 |
| c | string型データをとり、文字列の最初の1バイトをバイナリ列に挿入します。 |
| b | number型データをとり、符号付きバイト数値としてバイナリ列に挿入します。 |
| B | number型データをとり、符号無しバイト数値としてバイナリ列に挿入します。 |
| h | number型データをとり、符号付き2バイト数値としてバイナリ列に挿入します。 |
| H | number型データをとり、符号無し2バイト数値としてバイナリ列に挿入します。 |
| i | number型データをとり、符号付き4バイト数値としてバイナリ列に挿入します。 |
| I | number型データをとり、符号無し4バイト数値としてバイナリ列に挿入します。 |
| l | number型データをとり、符号付き4バイト数値としてバイナリ列に挿入します。 |
| L | number型データをとり、符号無し4バイト数値としてバイナリ列に挿入します。 |
| q | number型データをとり、符号付き8バイト数値としてバイナリ列に挿入します。 |
| Q | number型データをとり、符号無し8バイト数値としてバイナリ列に挿入します。 |
| f | number型データをとり、float数値 (4バイト) としてバイナリ列に挿入します。 |
| d | number型データをとり、double数値 (8バイト) としてバイナリ列に挿入します。 |
| s | string型データをとり、指定の文字エンコードに変換してバイナリ列に挿入します。文字エンコード名は、format中にブレース記号 "{" および "}" で囲んで指定します。この指定子の場合、先行する個数を表す数値は、変換した結果からバイナリ列に挿入するバイト数になります。 |

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

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

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

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#len()  
バイト数を返します。

binary#pointer(offset:number => 0)  
t.b.d.

binary#reader() {block?}  
読み込み用ストリームに変換した結果を返します。

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

binary#unpack(format:string, offset:number => 0)  
引数fomatで指定したフォーマットに基づいて、バイナリデータから数値や文字列を抽出し、その結果をリストで返します。引数offsetは、抽出する位置をバイト単位で指定します。指定した位置がバイナリデータの範囲外になるとエラーになります。  
format中には、データの個数を表す数値に続いて以下の指定子を記述します。

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

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

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

binary#unpacks(format:string, offset:number => 0, cnt?:number) {block?}  
引数fomatで指定したフォーマットに基づいて、バイナリデータから数値や文字列を抽出するイテレータを返します。引数formatとoffsetの意味はbinary#unpackと同じです。cntは抽出する回数を指定し、これが省略されるとバイナリデータの終端まで抽出を続けます。  
blockをつけると、データを抽出するごとにその内容を評価します。ブロックパラメータの形式は |elems[], idx:number| で、elemsに抽出結果、idxに0から始まるインデクス番号が入ります。

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

codecクラス

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。

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。

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 |

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。

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。

colorクラス

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。

インスタンスの生成

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

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

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 |  | 名前 | RGB値 |
| black | 0, 0, 0 |  | silver | 192, 192, 192 |
| maroon | 128, 0, 0 |  | red | 255, 0, 0 |
| green | 0, 128, 0 |  | lime | 0, 255, 0 |
| olive | 128, 128, 0 |  | yellow | 255, 255, 0 |
| navy | 0, 0, 128 |  | blue | 0, 0, 255 |
| purple | 128, 0, 128 |  | fuchsia | 255, 0, 255 |
| teal | 0, 128, 128 |  | aqua | 0, 255, 255 |
| gray | 128, 128, 128 |  | white | 255, 255, 255 |

クラスプロパティ

|  |  |  |  |
| --- | --- | --- | --- |
| プロパティ | 型 | R/W | 説明 |
| names | string | R | カラー名の一覧が格納されています |
| Black | color | R | 色要素 #000000を持ったcolorインスタンスです |
| Maroon | color | R | 色要素 #800000を持ったcolorインスタンスです |
| Green | color | R | 色要素 #008000を持ったcolorインスタンスです |
| Olive | color | R | 色要素 #808000を持ったcolorインスタンスです |
| Navy | color | R | 色要素 #000080を持ったcolorインスタンスです |
| Purple | color | R | 色要素 #800080を持ったcolorインスタンスです |
| Teal | color | R | 色要素 #008080を持ったcolorインスタンスです |
| Gray | color | R | 色要素 #808080を持ったcolorインスタンスです |
| Silver | color | R | 色要素 #c0c0c0を持ったcolorインスタンスです |
| Red | color | R | 色要素 #ff0000を持ったcolorインスタンスです |
| Lime | color | R | 色要素 #00ff00を持ったcolorインスタンスです |
| Yellow | color | R | 色要素 #ffff00を持ったcolorインスタンスです |
| Blue | color | R | 色要素 #0000ffを持ったcolorインスタンスです |
| Fuchsia | color | R | 色要素 #ff00ffを持ったcolorインスタンスです |
| Aqua | color | R | 色要素 #00ffffを持ったcolorインスタンスです |
| White | color | R | 色要素 #ffffffを持ったcolorインスタンスです |

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 |

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。

### 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

dictクラス

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, ..} |

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。

environmentクラス

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。

インスタンスメソッド

environment#eval(expr:expr):map  
environmentのスコープ内でexprの内容を評価します。

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

errorクラス

インスタンスの生成

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

インスタンスメソッド

t.b.d

exprクラス

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。

### Expr要素と判定メソッド

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

|  |  |
| --- | --- |
| Expr要素 | 判定メソッド |
| Assign | expr#isassign() |
| Binary | expr#isbinary() |
| BinaryOp | expr#isbinaryop() |
| Block | expr#isblock() |
| BlockParam | expr#isblockparam() |
| Caller | expr#iscaller() |
| Container | expr#iscontainer() |
| DictAssign | expr#isdictassign() |
| Field | expr#isfield() |
| Force | expr#isforce() |
| Indexer | expr#isindexer() |
| Lister | expr#islister() |
| Prefix | expr#isprefix() |
| Quote | expr#isquote() |
| String | expr#isstring() |
| Suffix | expr#issuffix() |
| Symbol | expr#issymbol() |
| Unary | expr#isunary() |
| UnaryOp | expr#isunaryop() |
| Value | expr#isvalue() |

要素を参照するインスタンスメソッド

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

expr#car()  
*compound*要素が持つcarの内容をexpr型で返します。

expr#cdr()  
*compound*要素が持つcdrの内容をexpr型で返します。

expr#child()  
*unary*要素が持つchildの内容をexpr型で返します。

expr#each() {block?}  
*container*要素が持つ子要素の内容をexpr型で返すイテレータを生成します。  
blockをつけると、子要素ごとにその内容を評価します。ブロックパラメータの形式は |expr:expr, idx:number| で、exprに子要素、idxに0から始まるインデクス番号が入ります。

expr#getstring()  
string要素の文字列データを返します。

expr#getsymbol()  
symbol要素のシンボル値を返します。

expr#getvalue()  
value要素の値を返します。

expr#left()  
*binary*要素の左側要素の内容をexpr型で返します。

expr#right()  
*binary*要素の右側要素の内容をexpr型で返します。

### その他のインスタンスメソッド

expr#eval()  
exprの内容を現在の環境で評価します。

expr#exprname()  
要素名を文字列で返します。

expr#tofunction(`args\*)  
指定した引数列を持つ関数に変換します。

expr#unquote()  
exprの内容がquoteされているとき、それを取り除きます。

### 式を構成する要素

*expr*

value

symbol

string

value: any

symbol: symbol

string:string

*expr*

*unary*

*binary*

*container*

*compound*

1

1

1

2

1

\*

lister

1

1

1

1

*unary*

unaryop

quote

force

prefix

suffix

*binary*

binaryop

assign

dictassign

prefix

*container*

root

block

blockparam

lister

*compound*

indexer

caller

lister

*container*

*expr*

1

\*

1

1

functionクラス

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。

インスタンスプロパティ

|  |  |  |  |
| --- | --- | --- | --- |
| プロパティ | 型 | R/W | 説明 |
| symbol | symbol | R/W | 関数の名前を文字列で返します |
| name | string | R/W | 関数のシンボルを返します |
| 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。 |
| args | list | R | 関数の引数リストで宣言されている変数シンボルのリストです |
| expr | expr | R | 関数の本体です |
| help | string | R/W | 関数のヘルプを返します。  文字列を代入すると、それを新たなヘルプとして登録します。 |

インスタンスメソッド

function#diff(var?:symbol)

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

関数の内容は以下の条件を満たしている必要があります。

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

imageクラス

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。

インスタンスメソッド

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

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

image#delpalette():reduce  
イメージに関連付けられたパレットを削除します。

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

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

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

image#extract(x:number, y:number, width:number, height:number, element:symbol, dst):void  
t.b.d.

image#fill(color:color):void  
イメージ全体を指定した色で塗りつぶします。

image#fillrect(x:number, y:number,  
 width:number, height:number, color:color):map:void  
指定の範囲を指定した色で塗りつぶします。

image#flip(orient:symbol):map

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

`horz 左右反転

`vert 上下反転

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

image#getpixel(x:number, y:number):map  
指定の位置の色データをcolor型で返します。

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

image#putpixel(x:number, y:number, color:color):map:void  
引数x、yで指定した位置のピクセル色データをcolorに変更します。

image#read(stream:stream, imgtype?:string):map:reduce  
引数streamからイメージデータを読み込みます。このメソッドを実行するイメージインスタンスは、バッファが未確保である必要があります。すでにバッファを持っていた場合はエラーになります。  
引数imgtypeには、"jpeg" や"png" というようにイメージタイプ名を文字列で指定します。この引数が省略されると、イメージファイルのヘッダ情報やファイル名のサフィックスからイメージタイプを識別します。

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

image#resize(width?:number, height?:number):map:[box]  
イメージを指定の大きさにリサイズしたイメージインスタンスを生成して返します。  
widthおよびheightにリサイズ結果の大きさを指定します。どちらかを省略した場合、オリジナルの縦横比率を保つようにリサイズされます。アトリビュート:boxを指定して、widthのみを指定すると、縦横がいずれもwidthの正方形が指定されます。

image#rotate(rotate:number, background?:color):map  
イメージを引数rotateで指定した角度だけ回転させたイメージインスタンスを生成して返します。rotateの数値はdegreeで表わし、正の数が時計回り、負で反時計回りになります。  
引数backgroundは、回転させたときにできる余白を塗りつぶす色を指定します。省略すると、黒で塗りつぶします。

image#setalpha(alpha:number, color?:color, alphaRest:number => 255):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の値に置き換えたイメージインスタンスを生成して返します。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colorに合致しないピクセルのアルファ値はalphaRestの値に置き換えます。

image#size()  
イメージの幅と高さをリストにして返します。

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

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

image#write(stream:stream, imgtype?:string):map:reduce  
引数streamにイメージデータを書き込みます。このメソッドを実行するイメージインスタンスは、バッファを持っている必要があります。バッファを持っていない場合はエラーになります。  
引数imgtypeには、"jpeg" や"png" というようにイメージタイプ名を文字列で指定します。この引数が省略されると、ストリームについているファイル名のサフィックスからイメージタイプを識別します。

iteratorクラス

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。

fill(n:number, value?) {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nに生成する個数、valueに値を指定します。valueには任意の型のデータを指定できます。valueは省略可能で、省略した場合はnil値を要素に持つイテレータになります。

rands(num?:number, range?:number) {block?}  
指定の数だけ乱数を出力するイテレータを生成します。  
引数numに生成する個数、rangeに値の上限値を整数で指定します。出力する数値xは0 < x < rangeを満たす整数値になります。numが省略されると、無限に乱数を出力します。rangeが省略されると、出力する数値xは0 < x < 1 を満たす小数値になります。

range(num:number, num\_end?:number, step?:number):map {block?}  
開始値と終了値、および間隔を指定して連続する数列を出力するイテレータを生成します。  
引数numのみを指定すると、0からnum – 1 までの整数を出力します。  
引数numと num\_end を指定すると、numから num\_end – 1 までの整数を出力します。  
引数num, num\_endおよびstepを指定すると、numを開始値にして、stepごとに数値をインクリメントしてnum\_endを超えない範囲までの数値を出力します。  
連続した数値を出力するのに、オペレータ ".." を使うこともできます。"n..m" という形式では、nからmまでの整数を出力するイテレータになります。また、"n.." と指定すると、nを始点にして、無限にインクリメントするイテレータになります。

interval(a:number, b:number, samples:number):map:[open,open\_l,open\_r] {block?}  
範囲とサンプル数を指定して数列を出力するイテレータを生成します。一般式は以下のとおりです。 引数aに最小値、bに最大値を指定すると、サンプル数samples個だけ、[a, b] の範囲内で等間隔な数列を出力します。  
アトリビュート:open, :open\_l, :open\_r を指定すると、範囲のオープン条件を指定できます。:openを指定すると、範囲指定が(a, b)に、:open\_lでは(a, b]、:open\_rでは[a, b)になります。

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。

listクラス

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。

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

list#count(criteria?) / iterator#count(criteria?)  
リストまたはイテレータ中、条件に合致する要素の数を返します。条件criteriaには値または関数を指定します。  
criteriaを省略すると、要素中で真値と判断できるものの数を返します。  
criteriaに値を指定した場合、その値と要素を比較し、等しいと判断したものの数を数えます。  
criteriaに渡す関数は、引数を一つとりboolean値を返すものを指定します。list#countメソッドは要素をひとつずつ関数に渡し、帰ってきたtrueの数を数えます。

list#each() {block?} / iterator#each() {block?}  
リストまたはイテレータの要素を順に走査するイテレータを返します。  
blockをつけると、要素ごとにその内容を評価します。ブロックパラメータの形式は |value, idx:number| で、valueに要素値、idxに0から始まるインデクス番号が入ります。

list#erase(idx\*:number):reduce  
引数idxで指定される位置の要素をリストから削除します。これは破壊的メソッドです。

list#filter(criteria) {block?} / iterator#filter(criteria) {block?}  
リストまたはイテレータ中、引数criteriaで指定した条件に合致する要素を返すイテレータを生成します。  
criteriaに関数を指定すると、各要素を引数にしてその関数を呼び出し、関数がtrueを返したときの要素を抽出します。  
criteriaにリストまたはイテレータを指定すると、criteria中でtrueとなる位置のデータを抽出します。  
blockをつけると、要素ごとにその内容を評価します。ブロックパラメータの形式は |value, idx:number| で、valueに要素値、idxに0から始まるインデクス番号が入ります。

list#find(criteria?):[index] / iterator#find(criteria?):[index]  
リストまたはイテレータ中、引数criteriaで指定した条件に合致する要素の値を返します。criteriaを省略すると、trueと判定される要素の値を返します。アトリビュートに:index を指定すると、要素の値ではなくインデクス値を返します。  
criteriaに指定した値の型によって、以下の判定処理を行います。

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

list#first()  
リストの最初の要素値を返します。

list#flat()  
入れ子になったリストをすべて一次元に展開したリストを返します。

list#fold(n:number, nstep?:number):[iteritem] {block?} / iterator#fold(n:number):[iteritem] {block?}  
リストまたはイテレータの要素からn個ずつ組にしたリストを返すイテレータを生成します。引数nstepを指定すると、次に抽出する要素の間隔を指定できます。nstepを省略すると、次の抽出位置はn個先になります。アトリビュート:iteritemをつけると、組にした結果をリストではなくイテレータで返します。  
blockをつけると、抽出したリストごとにその内容を評価します。ブロックパラメータの形式は |elements:list, idx:number| で、elementsに抽出リスト、idxに0から始まるインデクス番号が入ります。

list#format(format:string):map / iterator#format(format:string) {block?}  
printf関数のフォーマットで、リストまたはイテレータの要素を文字列に変換します。

list#get(index:number):map:flat  
リスト中、引数indexで指定した位置にあるデータを取得します。

list#head(n:number):map {block?} / iterator#head(n:number):map {block?}  
リストまたはイテレータの最初のn個のデータを返すイテレータを生成します。  
blockをつけると、データごとにその内容を評価します。ブロックパラメータの形式は |value, idx:number| で、valueにデータ値、idxに0から始まるインデクス番号が入ります。

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

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

list#join(sep:string => "") / iterator#join(sep?:string)  
リストまたはイテレータの要素を文字列に変換し、それらを指定の文字列sepで連結します。

iterator#joinb()  
イテレータで返されるbinary型の要素を連結した結果を返します。要素がbinary型でない場合、エラーになります。

list#last()  
リストの最後の要素を返します。

list#len() / iterator#len()  
リストまたはイテレータの要素の数を返します。

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

list#max():[index,last\_index,indices] / iterator#max():[index,last\_index,indices]  
アトリビュートを何もつけずに実行したとき、リストまたはイテレータの要素のうち、大小比較をした結果が最も大きかった値を返します。  
アトリビュート:indexをつけると、最も大きな値が最初に見つかったインデクスを返します。アトリビュート:last\_indexでは、最も大きな値が最後に見つかったインデクスを返します。  
アトリビュート:indicesをつけると、最も大きな値が複数あった場合、それらすべてのインデクス値をリストにして返します。

list#min():[index,last\_index,indices] / iterator#min():[index,last\_index,indices]  
アトリビュートを何もつけずに実行したとき、リストまたはイテレータの要素のうち、大小比較をした結果が最も小さかった値を返します。  
アトリビュート:indexをつけると、最も小さな値が最初に見つかったインデクスを返します。アトリビュート:last\_indexでは、最も小さな値が最後に見つかったインデクスを返します。  
アトリビュート:indicesをつけると、最も小さな値が複数あった場合、それらすべてのインデクス値をリストにして返します。

list#nilto(replace) / iterator#nilto(replace)  
リストまたはイテレータの要素がnilのとき、指定した値に変換します。

list#offset(n:number):map {block?} / iterator#offset(n:number) {block?}  
リストまたはイテレータの先頭から指定の数だけ除外した後の要素を返すイテレータを生成します。

list#or() / iterator#or()  
要素間をオペレータ "|" で演算した結果を返します。

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

list#permutation(n?:number) {block?}  
リストから、重複しないn個のデータの順列組み合わせをリストにして返すイテレータを生成します。イテレータの要素数は、リストのデータ数をm個としたときmPnになります。  
blockをつけると、順列組み合わせのリストごとにその内容を評価します。ブロックパラメータの形式は |elements:list, idx:number| で、elementsに順列組み合わせリスト、idxに0から始まるインデクス番号が入ります。

list#pingpong(n?:number):[sticky,sticky\_l,sticky\_r] {block?} /  
iterator#pingpong(n?:number):[sticky,sticky\_l,sticky\_r] {block?}  
リストまたはイテレータの要素を順に走査し、最後に到達したら逆向きに走査、再び最初に戻ったら順に走査を繰り返すイテレータを生成します。  
引数nに走査結果で得られる要素の数を指定します。この引数を省略すると、無限に走査をくりかえす無限イテレータになります。アトリビュート:sticky, :sticky\_l, :sticky\_rは先頭または終端で折り返しをするときに要素を2度繰り返すか否かを指定します。:sticky\_lが先頭要素、:sticky\_rが終端要素、:stickyが両端の要素に対する繰り返し指定になります。

iterator#print(stream?:stream:w)  
要素の値を文字列にしてstreamに出力します。streamを省略した場合、標準出力に出力します。

list#printf(format:string, stream?:stream:w):void /  
 iterator#printf(format:string, stream?:stream:w)  
要素の値をprintfのフォーマットに従って文字列にし、streamに出力します。streamを省略した場合、標準出力に出力します。

iterator#println(stream?:stream:w)  
要素の値と改行をstreamに出力します。

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

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

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

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

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

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

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

list#reduce(accum) {block} / iterator#reduce(accum) {block}  
要素に対し畳み込み操作を行います。関数を実行すると、リストまたはイテレータから要素をひとつ受取り、この値valueと累積結果accumの値を|value, accum| というブロックパラメータの形式でブロックに渡します。ブロックの評価結果を新たなaccumとし、以下同じ操作を要素ごとに繰り返します。

list#replace(value, replace) / iterator#replace(value, replace)  
要素がvalueに等しいとき、replaceに置き換えるイテレータを生成します。

list#reverse() {block?} / iterator#reverse() {block?}  
要素列を逆から走査するイテレータを生成します。

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

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

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

list#shuffle():reduce  
リスト要素の順番を乱数で入れ替えた結果をリストにして返します。

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

list#skip(n:number):map {block?} / iterator#skip(n:number) {block?}  
指定数だけ要素を除外しながら要素列を走査するイテレータを返します。引数nに除外する要素数を指定します。

list#skipnil() {block?} / iterator#skipnil() {block?}  
nil要素をとりのぞくイテレータを生成します。

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

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

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

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

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

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

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

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

list#stddev() / iterator#stddev()  
要素から標準偏差を算出し、結果を返します。

list#sum() / iterator#sum()  
すべての要素を加算した結果を返します。

list#tail(n:number):map {block?} / iterator#tail(n:number) {block?}  
リストまたはイテレータの最後から指定の数の要素だけ返すイテレータを生成します。

list#variance() / iterator#variance()  
要素から分散値を算出し、結果を返します。

list#while (criteria) {block?}  
リストまたはイテレータから、条件に合致している間の要素を抽出するイテレータを生成します。  
criteria には関数またはイテレータを指定できます。  
関数は、一つの引数をとりboolean値を返すものを指定します。while関数はリストまたはイテレータの要素をひとつずつ関数に渡し、その戻り値がtrueの間だけ要素を抽出します。falseになったら処理を終了します。  
criteriaにイテレータを指定すると、while関数は抽出対象のリストまたはイテレータと同時にcriteriaのイテレータを走査し、これがtrue値の間だけ要素を抽出します。falseになったら処理を終了します。

matrixクラス

インスタンスの生成

matrix(nrows:number, ncols:number, value?)  
指定のサイズをもつmatirx型インスタンスを生成します。引数nrowsに行数、ncolsに桁数を指定します。引数valueに要素の値を指定します。valueを省略すると、要素の値はnilになります。

@@{block}

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

|  |  |
| --- | --- |
| スクリプト一般式 | 生成されるマトリクス |
| @@{{a11, a12, a13, …}, {a21, a22, a23, …}, …} | a11 a12 a13 …  a21 a22 a23 …  : |
| @@{a11, a21, a31, …} | a11  a21  : |
| @@{{a11, a12, a13, …}} | a11 a12 a13 … |

### 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。

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。

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m11, m12, m13, m21, m22, m23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m11, m21, m12, m22, m13, m23となります。

matrix#eachcol()  
列要素をリストにして返すイテレータを生成します。

matrix#eachrow()  
行要素をリストにして返すイテレータを生成します。

matrix#invert()  
逆行列を計算し、結果を返します。

matrix#issquare()  
正方行列のときtrue、それ以外はfalseを返します。

matrix#row(row:number):map  
指定の行の要素をリストにして返します。

matrix#rowsize()  
マトリクスの行の数を返します。

matrix#set(value)  
すべての要素をvalueで置き換えます。

matrix#setcol(col:number, value)  
指定の列の要素をvalueで置き換えます。

matrix#setrow(row:number, value)  
指定の行の要素をvalueで置き換えます。

matrix#submat(row:number, col:number, nrows:number, ncols:number):map  
部分行列を返します。

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

matrix#transpose()  
転置行列を返します。

matrix#roundoff(threshold:number => 1e-10)  
指定した値ですべての要素を丸めます。

paletteクラス

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。

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。

semaphoreクラス

インスタンスの生成

semaphore()  
semaphore型インスタンスを生成します。

インスタンスメソッド

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

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

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

streamクラス

インスタンスの生成

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

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

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

|  |  |
| --- | --- |
| r | 読み込みモード |
| w | 書き込みモード |
| a | 追加書き込みモード |

引数modeを省略し、パス名の先頭に ">" をつけると、書き込みモードでストリームをオープンします。

引数codecは、ストリームを文字列として読み書きするときのコーデックを指定します。この指定は、ストリームをバイナリデータとして扱うメソッドや関数には影響しません。

### ストリーム操作を行うグローバル関数

copy(src:stream:r, dst:stream:w, bytesunit:number => 65536):map:void {block?}  
入力用ストリームsrcから出力用ストリームdstにデータをコピーします。コピー処理はデフォルトで65536バイト単位で行われますが、引数bytesunitでこれを変更することができます。  
blockを指定すると、コピー単位ごとにブロックパラメータを|buff:binary| という形式でブロックに渡して評価します。buffは入力用ストリームから入力したデータが入り、評価結果がbinary型の値であればそれを出力用ストリームに出力します。それ以外の型の場合はもとのデータを出力します。

template(src:stream:r, dst?:stream:w):map:[lasteol,noindent] {block?}  
入力用ストリームsrcからテンプレート文字列を読み込み、スクリプトの評価結果を埋め込んだ文字列を出力用ストリームdstに出力します。引数dstを省略すると、結果はstring型のデータで戻り値として返されます。  
スクリプトはテキスト中に "${" と "}" に囲まれた領域に記述されます。  
スクリプトの評価結果で、最後に現れた改行コードはとりのぞかれます。アトリビュート:lasteolをつけると、この改行コードをとりのぞかずに出力に含めます。  
スクリプトの出力結果が複数行にわたるとき、スクリプトの開始を表す "${" の行の先頭にある空白文字が各行に追加されます。アトリビュート :noindent をつけると、このインデンテーション機能を無効にします。

readlines(stream?:stream:r):[chop] {block?}  
入力用ストリームstreamからテキストを読み込み、行ごとに分割した文字列を返すイテレータを生成します。デフォルトでは、改行記号まで含めた文字列を返しますが、アトリビュート　:chop をつけると、改行記号を削除します。  
引数streamを省略すると、標準入力からテキストを読み込みます。  
blockをつけると、行ごとにその内容を評価します。ブロックパラメータの形式は |line:string, idx:number| で、lineに行ごとの文字列、idxに0から始まるインデクス番号が入ります。

インスタンスメソッド

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

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

stream#delcr(flag?:boolean):reduce

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

stream#addcr(flag?:boolean):reduce

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

stream#flush():void  
ストリームへの出力内容をフラッシュします。

stream#peek(len?:number)  
ストリームからデータをlenバイトだけ先読みします。シーク位置は変化しません。

stream#print(values\*):map:void  
引数に指定した値の内容を文字列にしてストリームに出力します。

stream#printf(format:string, values\*):map:void  
printfのフォーマットに基づいてストリームに出力します。

stream#println(values\*):map:void  
引数に指定した値の内容を文字列にしてストリームに出力します。最後に改行コードをストリームに出力します。

stream#read(len?:number)  
ストリームからデータをlenバイトだけ読み込みます。シーク位置もlenだけ先に進みます。

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

stream#readlines(nlines?:number):[chop] {block?}  
入力用ストリームstreamからテキストを読み込み、行ごとに分割した文字列を返すイテレータを生成します。デフォルトでは、改行記号まで含めた文字列を返しますが、アトリビュート　:chop をつけると、改行記号を削除します。  
blockをつけると、行ごとにその内容を評価します。ブロックパラメータの形式は |line:string, idx:number| で、lineに行ごとの文字列、idxに0から始まるインデクス番号が入ります。

stream#readtext()  
ストリームからテキストをすべて読み取り、文字列にして返します。

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

stream#seek(offset:number, origin?:symbol):reduce  
ストリームのシーク位置を先頭からoffsetの位置に動かします。originに`curを設定すると、現在のシーク位置にoffsetだけ足した位置に移動します。このとき、offsetにはマイナスの値を設定することができます。

stream#setencoding(encoding:string, dos\_flag?:boolean)  
ストリームのエンコーディングを指定します。引数encodingは、ストリームを文字列として読み書きするときの文字エンコーディング名です。dos\_flagをtrueに設定すると、改行コードを以下のように変換します。  
入力時: CR-LF をLFに変換  
出力時: LF をCR-LFに変換

stream#tell()  
ストリームのシーク位置を返します。

stream#write(buff:binary):reduce  
ストリームにバイナリデータを書き込みます。

stream#copyfrom(src:stream:r, bytesunit:number => 65536):map:reduce {block?}  
入力用ストリームsrcからストリームインスタンスにデータをコピーします。コピー処理はデフォルトで65536バイト単位で行われますが、引数bytesunitでこれを変更することができます。  
blockを指定すると、コピー単位ごとにブロックパラメータを|buff:binary| という形式でブロックに渡して評価します。buffは入力用ストリームから入力したデータが入り、評価結果がbinary型の値であればそれを出力用ストリームに出力します。それ以外の型の場合はもとのデータを出力します。

stream#copyto(dst:stream:w, bytesunit:number => 65536):map:reduce {block?}  
ストリームインスタンスから出力用ストリームdstにデータをコピーします。コピー処理はデフォルトで65536バイト単位で行われますが、引数bytesunitでこれを変更することができます。  
blockを指定すると、コピー単位ごとにブロックパラメータを|buff:binary| という形式でブロックに渡して評価します。buffは入力用ストリームから入力したデータが入り、評価結果がbinary型の値であればそれを出力用ストリームに出力します。それ以外の型の場合はもとのデータを出力します。

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。 |

stringクラス

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。

インスタンスメソッド

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

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

string#chop(suffix:\*string):[icase,eol]  
何も引数やアトリビュートをつけずに実行すると、文字列中の最後の一文字を取り除いた結果を返します。  
アトリビュート:eolをつけると、最後が改行記号のときのみ取り除きます。コードがCR-LFという連なりになっている場合は、これら2文字をとりのぞきます。  
引数に文字列を指定すると、これらの文字列が最後にあらわれたときのみとりのぞきます。この文字列は複数指定することができます。アトリビュート:icaseが指定されると、大文字と小文字を区別しません。また、アトリビュート:eolとともに実行した場合は、まず改行コードがあればそれをとりのぞき、その後指定文字列の除去を行います。

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

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

string#encode(codec:codec)  
文字列を指定のコーデックで変換した結果をbinary型として返します。

string#endswith(suffix:string, endpos?:number):map:[rest,icase]  
文字列がsuffixで終了している場合はtrue、それ以外はfalseを返します。アトリビュート:icaseをつけると、大文字と小文字を区別しません。  
アトリビュート:restをつけると、文字列がsuffixで終了している場合、それよりも前の文字列を返します。それ以外はnilを返します。

string#escapehtml()  
HTML書式で処理ができるようにした文字列を返します。

string#escapeuri()  
URI書式で処理ができるようにした文字列を返します。

string#template(stream?:stream:w):[lasteol,noindent]  
文字列中に記述されたスクリプトを評価し、文字列に挿入します。スクリプトは文字列中で "${" と "}" に挟んで記述します。

string#find(sub:string, pos:number => 0):map:[rev,icase]  
文字列subが見つかった文字位置を返します。見つからない場合はnilを返します。引数posを指定すると、その位置から文字列を探します。アトリビュート :rev をつけると、後尾から文字列を探します。アトリビュート:icaseをつけると、大文字と小文字の区別をつけません。

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

string#isempty()  
文字列が空のときtrue、それ以外はfalseを返します。

string#left(len?:number):map  
左から指定文字数だけ取り出した文字列を返します。

string#len()  
文字列中の*文字数* を返します。バイト数でないことに注意してください。

string#lower()  
アルファベットを小文字に変換した結果を返します。

string#mid(pos:number => 0, len?:number):map  
引数posの位置から長さlen文字数だけ取り出した文字列を返します。

string#print(stream?:stream:w):void  
文字列を引数streamで指定したストリームに出力します。

string#println(stream?:stream:w):void  
文字列を引数streamで指定したストリームに出力し、最後に改行を出力します。

string#reader() {block?}  
文字列内の文字コードを1バイトずつとりだすストリームを返します。文字コードはUTF-8です。

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

string#right(len?:number):map  
右から指定文字数だけ取り出した文字列を返します。

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

string#startswith(prefix:string, pos:number => 0):map:[rest,icase]  
文字列がprefixで始まっている場合はtrue、それ以外はfalseを返します。アトリビュート:icaseをつけると、大文字と小文字を区別しません。  
アトリビュート:restをつけると、文字列がsuffixで始まっている場合、それよりも後の文字列を返します。それ以外はnilを返します。

string#strip():[both,left,right]  
文字列の左右にある空白や改行要素をとりのぞいた結果を返します。  
アトリビュート:leftをつけると、左側のみの空白・改行要素をとりのぞきます。アトリビュート:rightをつけると、右側のみとりのぞきます。アトリビュート:bothは両側の空白・改行要素をとりのぞき、これがデフォルトの動作になります。

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

string#unescapeuri()  
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#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。

sysモジュール

## 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。

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。

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 |

fsモジュール

## 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。

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。

## パスのサーチ

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

モジュール関数

fs.chdir(pathname:string) {block?}  
カレントディレクトリを設定します。ブロックを指定した場合、カレントディレクトリを設定した後にブロックの内容を評価し、元のディレクトリに戻ります。

fs.getcwd()  
カレントディレクトリを取得します。

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

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

fs.remove(pathname:string):map:void  
ファイルを削除します。

fs.rename(src:string, dst:string):map:void  
ファイルまたはディレクトリの名前を変更します。

fs.chmod(mode:number, pathname:string):void  
ファイルの属性を数値で設定します。属性は数値のビット位置に対応しており、ビットを1に設定するとその属性が有効になります。ビット位置と属性の関係を以下に示します。  
 b8 b7 b6 所有者のリード、ライト、実行属性  
 b5 b4 b3 グループのリード、ライト、実行属性  
 b2 b1 b0 その他のユーザのリード、ライト、実行属性

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

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

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

最初に、属性を設定する対象を指定し、続けて設定方法と属性を指定します。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 設定する対象 | |  | 設定方法 | |  | 属性 | |
| u | 所有者 |  | - | 属性をとりのぞく |  | r | リード属性 |
| g | グループ |  | + | 属性を加える |  | w | ライト属性 |
| o | その他のユーザ |  | = | 属性を設定する |  | x | 実行属性 |
| a | 全てのユーザ |  |

## fs.statクラス

### 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 |

osモジュール

## 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。

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。 |

モジュール関数

os.exec(pathname:string, args\*:string):map:[fork]  
外部実行可能ファイルを実行します。引数pathnameに実行可能ファイルのファイル名、argsに引数を指定します。  
デフォルトでは、この関数は実行可能ファイルが終了するのを待ち、実行結果のエラーレベル (C言語のプログラムならばmain関数の戻り値またはexit関数の引数値) を戻り値として返します。このとき、標準出力および標準エラー出力の内容をそれぞれos.stdoutとos.stderrに指定したストリームに対して出力します。  
アトリビュート:forkをつけると、実行可能ファイルを起動した後、すぐに関数から処理が戻ります。この場合、戻り値は常に0です。

os.fromnative(buff:binary):map  
OS依存の文字列をスクリプトで処理できる文字列に変換します。

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

os.putenv(name:string, value:string):void  
引数nameに対応する環境変数の値をvalueに設定します。

os.redirect(stdin:stream:nil:r, stdout:stream:nil:w,  
 stderr?:stream:w) {block?}  
標準入力os.stdin、標準出力os.stdoutおよび標準エラー出力os.stderrを指定のstreamインスタンスに設定します。引数stderrは省略可能で、省略するとstdoutに指定したのと同じstreamインスタンスに設定します。  
stdinにnilを設定すると、標準入力に何も接続しません。stdoutやstderrにnilを設定すると、これらの出力を抑止することができます。  
ブロックを指定して実行すると、ストリームを設定してからブロックを評価し、評価後に設定をもとにもどします。  
os.redirectの戻り値はブロックが指定されている場合はその評価値、指定されていなければ常にnilです。

os.tonative(str:string):map  
スクリプトで処理できる文字列からOS依存の文字列に変換します。

pathモジュール

## 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。

モジュール関数

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

path.bottom(pathname:string):map  
パス名をパスセパレータで区切った時の最後の要素名を返します。

path.cutbottom(pathname:string):map  
パス名をパスセパレータで区切った時の最後の要素名を取り除いた結果を返します。

path.dir(pathname?:string, pattern\*:string):map:flat:[stat,icase,file,dir] {block?}  
ディレクトリを表すパス名を指定し、含まれるファイルまたはディレクトリをサーチします。  
引数pathnameはパス名です。引数patternには、ファイルまたはディレクトリのベース名に対するパターンを0個以上指定します。この引数を省略すると、すべてのファイルまたはディレクトリをサーチします。  
アトリビュート:statをつけるとパス名ではなく詳細情報を含んだstat型オブジェクトを返します。:icaseは、パターンマッチングの際に大文字と小文字の区別をなくすアトリビュートです。:fileや:dirをつけると、サーチ対象をそれぞれファイルまたはディレクトリに限定できます。  
ブロック式をつけると、各サーチ結果ごとにブロックが繰り返し評価されます。このとき、ブロックには|pathname:string, idx:number| という形式で引数が渡されます。pathnameはサーチ結果のパス名、idxはループのインデクス番号です。

path.dirname(pathname:string):map  
パス名からディレクトリ名要素を抽出します。

path.exists(pathname:string):map  
指定したパスが存在するときtrueを返します。それ以外のときはfalseを返します。

path.filename(pathname:string):map  
パス名からファイル名要素を抽出します。

path.glob(pattern:string):map:flat:[stat,icase,file,dir] {block?}  
パターンに適合するファイルやディレクトリをサーチします。  
引数patternにパターンを指定します。このパターンはディレクトリ名を含むことができ、パス名の途中のディレクトリ名にもワイルドカードを使えます。  
アトリビュート:statをつけるとパス名ではなく詳細情報を含んだstat型オブジェクトを返します。:icaseは、パターンマッチングの際に大文字と小文字の区別をなくすアトリビュートです。:fileや:dirをつけると、サーチ対象をそれぞれファイルまたはディレクトリに限定できます。  
ブロック式をつけると、各サーチ結果ごとにブロックが繰り返し評価されます。このとき、ブロックには|pathname:string, idx:number| という形式で引数が渡されます。pathnameはサーチ結果のパス名、idxはループのインデクス番号です。

path.join(paths+:string):map:[uri]  
パス名をつなぎあわせた結果を返します。  
つなぎあわせるときのパスセパレータは、現在動作しているOSがWindows系の場合はバックスラッシュ "\"、それ以外の場合はスラッシュ "/" を使用します。ただし、アトリビュート:uriを指定するとパスセパレータとして常にスラッシュ "/" を使用します。

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

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

マッチングパターンには以下のワイルドカードを使用することができます。

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

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

パス名を以下の条件に従って整形します。

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

path.split(pathname:string):map:[bottom]  
パス名をディレクトリ名とファイル名に分離し、リストにして返します。これは、path.dirnameとpath.filenameの結果をあわせたものと同じです。  
アトリビュート:bottomをつけると、パスセパレータで区切った時の前の要素と最後の要素をリストにして返します。これは、path.cutbottomとpath.bottomの結果をあわせたものと同じです。

path.splitext(pathname:string):map  
パス名のサフィックスを分離し、分離した前の部分とサフィックスをリストにして返します。

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

path.walk(pathname?:string, maxdepth?:number, pattern\*:string)  
 :map:flat:[stat,icase,file,dir] {block?}  
パス名で指定したディレクトリを基点として含まれるファイルまたはディレクトリを再帰的にサーチします。  
引数pathnameはパス名です。maxdepthには、サーチするディレクトリの深さを指定します。0を指定すると基点のディレクトリのみのサーチとなり、これはpath.dirの動作と同じになります。省略すると、深さの制限がなくなります。  
引数patternには、ファイルまたはディレクトリのベース名に対するパターンを0個以上指定します。この引数を省略すると、すべてのファイルまたはディレクトリをサーチします。  
アトリビュート:statをつけるとパス名ではなく詳細情報を含んだstat型オブジェクトを返します。:icaseは、パターンマッチングの際に大文字と小文字の区別をなくすアトリビュートです。:fileや:dirをつけると、サーチ対象をそれぞれファイルまたはディレクトリに限定できます。  
ブロック式をつけると、各サーチ結果ごとにブロックが繰り返し評価されます。このとき、ブロックには|pathname:string, idx:number| という形式で引数が渡されます。pathnameはサーチ結果のパス名、idxはループのインデクス番号です。

mathモジュール

## 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。

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 、負のとき になります。アトリビュート:degをつけると、degree値で結果を返します。

math.bezier(nums[]+:number)

math.ceil(num):map  
小数点以下一位を切り上げた数値を返します。

math.conj(num):map  
共役複素数を返します。

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

math.cosh(num):map  
ハイパボリックコサインを計算します。指定する角度の単位はラジアンです。

math.covariance(a:iterator, b:iterator)  
二つのイテレータ要素間の共分散値を計算します。

math.cross\_product(a[], b[])

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

math.dot\_product(a[], b[])

math.exp(num):map  
底が 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 のlog値を計算します。

math.log10(num):map  
底が 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。

timeモジュール

## 概要

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

モジュール関数

time.clock()  
1/100秒ごとに数値が増えるシステムクロック値を返します。

time.isleap(year:number):map  
指定した年がうるう年のときtrueを返します。それ以外はfalseを返します。

time.monthdays(year:number, month:number):map  
西暦と月を受け取り、その月の最終日を返します。

time.parse(str:string):map

time.sleep(secs:number)  
指定した秒数だけスリープします。

time.weekday(year:number, month:number, day:number):map  
指定した日の曜日をインデクス値で返します。日曜日が0で土曜日が6になります。

モジュール変数

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

datetimeクラス

### 概要

時刻を表すクラスです。

インスタンスの生成

time.datetime(year:number => 0, month:number => 1, day:number => 1,  
 hour:number => 0, min:number => 0, sec:number => 0,  
 usec:number => 0, minsoff?:number):map  
年月日および時刻を指定したdatetimeインスタンスを生成します。

time.now():[utc]  
現在の年月日および時刻が入ったdatetimeインスタンスを生成します。

time.time(hour:number => 0, minute:number => 0,  
 sec:number => 0, usec:number => 0):map  
時刻を設定したdatetimeインスタンスを生成します。年月日は0年1月1日に設定されます。

time.today():[utc]  
今日の日付が入ったdatetimeインスタンスを生成します。時刻は00:00:00が設定されます。

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。

### 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 |

timedeltaクラス

### 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

インスタンスの生成

time.delta(days:number => 0, secs:number => 0, usecs:number => 0)  
指定した値を持つtimedeltaインスタンスを生成します。

### 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。 |

conioモジュール

## 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。

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

hashモジュール

## 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。

モジュール関数

hash.crc32(stream?:stream:r) {block?}  
CRC32値を算出するストリームを作成して返します。引数streamを指定すると、そのデータ内容を生成したストリームに書き込みます。

hash.md5(stream?:stream:r) {block?}  
MD5値を算出するストリームを作成して返します。引数streamを指定すると、そのデータ内容を生成したストリームに書き込みます。

hash.sha1(stream?:stream:r) {block?}  
SHA1値を算出するストリームを作成して返します。引数streamを指定すると、そのデータ内容を生成したストリームに書き込みます。

httpモジュール

## 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/

## 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。

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 |

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。

## http.serverクラス

### 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。

### 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 |

### 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。

### サンプルプログラム

サーバプログラムの例を以下に示します。

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

## http.clientクラス

### 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。

### 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。

### 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' |

### サンプルプログラム

クライアントプログラムの例を以下に示します。

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

## http.statクラス

### 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。

### 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。

### 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 |

### 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。

## http.requestクラス

### 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。

### 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。

### 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.splituri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 |

### 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。

## http.sessionクラス

### 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。

### 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 |

## http.responseクラス

### 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。

### 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。

### 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 |

### 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。

bmpモジュール

## 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。

## 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)

## imageクラスの拡張

### 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。

# gifモジュール

## 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

## 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)

## gif.contentクラス

### 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。

### 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。

GIF Data Stream

Extension Block

Image Descriptor

Header

Logical Screen Descriptor

Global Color Table

Graphic Control Extension

Comment Extension

Plain Text Extension

Application Extension

Image Descriptor

Local Color Table

Trailer

Image Data

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。

### 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。

### 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。

### 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。

### 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。

### 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 |

## imageクラスの拡張

### 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。

### 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 |

### 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。 |

### 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。

# jpegモジュール

## 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/

## 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)

## jpeg.exifクラス

### 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。

### 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。

### 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。 |

### 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。

## jpeg.ifdクラス

### 概要

### インスタンスの生成

### インスタンスプロパティ

### 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。

## jpeg.tagクラス

### 概要

### インスタンスの生成

### インスタンスプロパティ

|  |  |  |  |
| --- | --- | --- | --- |
| プロパティ | データ型 | R/W | 内容 |
| id | number | R |  |
| name | string | R |  |
| symbol | symbol | R |  |
| type | number | R |  |
| value | any | R |  |
| ifd | jpeg.ifd | R |  |

## imageクラスの拡張

### 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。

# msicoモジュール

## 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

## 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)

## msico.contentクラス

### 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。

### 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。

### 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。

## imageクラスの拡張

### 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。

# pngモジュール

## 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

## 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)

## imageクラスの拡張

### 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。

# ppmモジュール

## 概要

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

モジュールの実装は以下のURLの記述に基づきます。

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

## 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)

## imageクラスの拡張

### 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。

# xpmモジュール

## 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。

## 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)

## imageクラスの拡張

### 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。

freetype モジュール

## 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/

## 関数

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。

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。

## freetype.faceクラス

### インスタンスメソッド

freetype.face#calcbbox(x:number, y:number, str:string):map  
t.b.d

freetype.face#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.face#cleardeco():reduce  
修飾要素をすべてとりのぞきます。

freetype.face#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。

freetype.face#setalpha(alpha:number):reduce  
描画時のアルファ要素を0から255までの数値で指定します。

freetype.face#setcolor(color:color):reduce  
描画時の色を指定します。

freetype.face#setheight(height:number):reduce  
フォントの高さをピクセル値で指定します。

freetype.face#setrotate(degree:number):reduce  
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。

freetype.face#setsize(width:number, height:number):reduce  
フォントの幅と高さをピクセル値で指定します。

freetype.face#setslant(slant:number):reduce  
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。

freetype.face#setstrength(strength:number):reduce  
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。

### 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 |  |
| face\_flag\_scalable | boolean | R | FT\_FACE\_FLAG\_SCALABLE |
| face\_flag\_fixed\_width | boolean | R | FT\_FACE\_FLAG\_FIXED\_WIDTH |
| face\_flag\_sfnt | boolean | R | FT\_FACE\_FLAG\_SFNT |
| face\_flag\_horizontal | boolean | R | FT\_FACE\_FLAG\_HORIZONTAL |
| face\_flag\_vertical | boolean | R | FT\_FACE\_FLAG\_VERTICAL |
| face\_flag\_kerning | boolean | R | FT\_FACE\_FLAG\_KERNING |
| face\_flag\_fast\_glyphs | boolean | R | FT\_FACE\_FLAG\_FAST\_GLYPHS |
| face\_flag\_multiple\_masters | boolean | R | FT\_FACE\_FLAG\_MULTIPLE\_MASTERS |
| face\_flag\_glyph\_names | boolean | R | FT\_FACE\_FLAG\_GLYPH\_NAMES |
| face\_flag\_external\_stream | boolean | R | FT\_FACE\_FLAG\_EXTERNAL\_STREAM |
| face\_flag\_hinter | boolean | R | FT\_FACE\_FLAG\_HINTER |
| face\_flag\_cid\_keyed |  | R | (未使用) |
| face\_flag\_tricky |  | R | (未使用) |
| style\_flag\_italic | boolean | R | FT\_STYLE\_FLAG\_ITALIC |
| style\_flag\_bold | boolean | R | FT\_STYLE\_FLAG\_BOLD |
| family\_name | string | R |  |
| style\_name | string | R |  |
| bbox | list | R |  |
| ascender | number | R |  |
| descender | number | R |  |
| height | number | R |  |

## imageクラスの拡張

### インスタンスメソッド

image#drawtext(face:freetype.face, 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。

sqlite3モジュール

## 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

## 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 |

## sqlite3.dbクラス

### 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。

### 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。

gzip モジュール

## 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/

## 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) |

## 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。

## streamクラスの拡張

### 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。

bzip2 モジュール

## 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/

## モジュール関数

bzip2.reader(stream:stream:r) {block?}  
ストリームstreamからbzip2形式で圧縮されたデータ列を読み込み、展開した結果を返すストリームを返します。

bzip2.writer(stream:stream:w) {block?}  
ストリームstreamにbzip2形式で圧縮したデータ列を書きこむストリームを返します。

## streamクラスの拡張

### インスタンスメソッド

stream#bzip2reader() {block?}  
ストリームstreamからbzip2形式で圧縮されたデータ列を読み込み、展開した結果を返すストリームを返します。

stream#bzip2writer() {block?}  
ストリームstreamにbzip2形式で圧縮したデータ列を書きこむストリームを返します。

zip モジュール

## 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

## 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。

## zip.readerクラス

### 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。

### 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。

## zip.writerクラス

### 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

### 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。

## zip.statクラス

### 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 |

tar モジュール

## 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。

## 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。

## 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 |

## tar.readerクラス

### 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

### 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。

## tar.writerクラス

### 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

### 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。

## tar.statクラス

### 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 |

reモジュール

## 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

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。

モジュール関数

re.match(pattern:re.pattern, str:string, pos:number => 0, endpos?:number):map  
正規表現 pattern に文字列 str がマッチしたときre.matchインスタンスを返します。マッチしない場合はnilを返します。  
引数posとendposでパターンをマッチさせる範囲を文字単位で指定することができます。endposは範囲に含める文字の次の文字位置を指定します。これらを省略した場合、文字列全体が処理対象になります。

re.scan(pattern:re.pattern, str:string, pos:number => 0, endpos?:number):map {block?}  
正規表現 pattern に文字列 str がマッチしたときre.matchインスタンスを返すイテレータを生成します。マッチすると、マッチした文字列の次の文字からパターンマッチングを行います。このようにして、パターンがマッチしなくなるまでre.matchインスタンスを返します。  
引数posとendposでパターンをマッチさせる範囲を文字単位で指定することができます。endposは範囲に含める文字の次の文字位置を指定します。これらを省略した場合、文字列全体が処理対象になります。

re.split(pattern:re.pattern, str:string, count?:number):map {block?}  
正規表現 pattern であらわされるパターンで文字列 str を分割します。引数countを指定すると、分割数をその数までに限定します。

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

文字列str中、正規表現patternにマッチする部分をreplaceで置換します。引数countを指定すると、置換する回数をその数までに限定します。

replaceには文字列または関数を指定します。

replaceに文字列を指定したとき、マッチ部分をその文字列で置き換えます。このとき、replace文字列中に "\0" という記述があったとき、この部分をマッチした全体の文字列で置き換えます。また、"\1"、"\2" … という記述はそれぞれマッチパターンのグループ1、グループ2…の文字列で置き換えます。

replaceに関数を渡したとき、関数を以下の形式で呼び出し、マッチ部分をこの関数の戻り値で置き換えます。

replace(strSub:string)

戻り値が文字列でない場合は、文字列に変換してから置き換えます。

re.matchクラス

### 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

### 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。

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 |

### 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。

re.patternクラス

### 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。

### 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。

stringクラスの拡張

### 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。

list/iteratorクラスの拡張

### 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+') |

csvモジュール

## 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。

## モジュール関数

csv.parse(str:string):map  
CSV形式のテキストを含んだ文字列を受け取り、カンマで区切られたフィールドの値を要素に持つリストを一行ずつ返すイテレータを生成します。

csv.reader(stream:stream:r) {block?}  
ストリームからCSV形式のテキストデータを読み込み、カンマで区切られたフィールドの値を要素に持つリストを一行ずつ返すイテレータを生成します。

csv.writer(stream:stream:w, format?:string) {block?}  
ストリームにCSV形式のテキストデータ出力するcsv.writerインスタンスを生成します。  
引数formatには数値データのフォーマット文字列を指定します。省略すると '%g' が使用されます。

## csv.writerクラス

### 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。 |

### 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。

## streamクラスの拡張

### インスタンスメソッド

stream#csvreader() {block?}  
ストリームからCSV形式のテキストデータを読み込み、カンマで区切られたフィールドの値を要素に持つリストを一行ずつ返すイテレータを生成します。

stream#csvwriter(format?:string) {block?}  
ストリームにCSV形式のテキストデータ出力するcsv.writerインスタンスを生成します。  
引数formatには数値データのフォーマット文字列を指定します。省略すると '%g' が使用されます。

xmlモジュール

## 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/

## モジュール関数

xml.read(stream:stream:r)

## xml.parserクラス

### インスタンスの生成

xml.parser() {block?}

### 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()

### インスタンスプロパティ

|  |  |  |  |
| --- | --- | --- | --- |
| プロパティ | データ型 | R/W | 説明 |
| children | xml.element[] | R/W |  |

### インスタンスメソッド

xml.parser#parse(stream:stream)

## xml.elementクラス

### インスタンスの生成

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

## streamクラスの拡張

### インスタンスメソッド

stream#xmlread()

# yamlモジュール

## 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/

## データオブジェクトの対応

## 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。

## streamクラスの拡張

### 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。

# uuid モジュール

## 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。

## 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。

mswinモジュール

## 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。

## mswin.oleクラス

### 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。

## mswin.regkeyクラス

### 概要

Windowsのレジストリを扱うクラスです。

### 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

### 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。

## COMについて

### COMサーバへの接続

COMはMicrosoftが開発したアプリケーションインターフェースの仕様です。Microsoft　WordやExcel、Internet ExplorerがCOMをサポートしており、これらのアプリケーションの動作をすべて外部からコントロールすることができます。このようなCOMを外部に提供しているアプリケーションやDLLをCOMサーバと呼びます。

mswin.oleでmswin.oleインスタンスを生成すると、指定したCOMサーバへの接続を確立し、COMサーバが提供するメソッドやプロパティを動的に作成します。

以下はMicrosoft Excelを起動し、既存のファイルをオープンする例です。

|  |
| --- |
| import(mswin)  mswin.ole('Excel.Application') {|app|  app.Visible = 1  app.Workbooks.Open(path.absname('hoge.xls'))  } |

以下はMicrosoft Wordを起動し、既存のファイルをオープンする例です。

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

### 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。

### 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。

### 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。

### イテレータの生成

イテレータを期待している文中にmswin.oleインスタンスを指定したとき、内包しているOLEオブジェクトがイテレータに対応していれば、適切なイテレータを生成します。以下は、Excelワークブック中の全てのワークシート名を表示する例です。

|  |
| --- |
| import(mswin)  mswin.ole('Excel.Application') {|app|  app.Visible = 1  wb = app.Workbooks.Open(path.absname('hoge.xlsx'))  for (ws in wb.WorkSheets) {  println(ws.Name)  }  } |

COMへのアクセスは、DispIDにDISPID\_NEWENUM を指定してDISPATCH\_METHODを実行しています。

# lets\_moduleモジュール

## 概要

バイナリモジュールのC++ソースコードとビルド用スクリプトのひな型を作成します。

|  |
| --- |
| $ **gura –i lets\_module hoge** |

# modbuildモジュール

## 概要

バイナリモジュールをビルドするためのモジュールです。使用するにはimport関数を使ってmodbuileモジュールをインポートします。

以下はModule\_hoge.cppからhoge.gurdをビルドするスクリプトの例です。

|  |
| --- |
| import(modbuild)  builder = modbuild.Builder()  builder.build('hoge', ['Module\_hoge.cpp']) |

## modbuild.Builderクラス

## 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 |

## 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。

# gurcbuildモジュール

## 概要

コンポジットファイルを作成するモジュールです。使用するにはimport関数を使ってgurdbuildモジュールをインポートします。

以下はコンポジットファイルhoge.gurcを作成するスクリプトの例です。

|  |
| --- |
| import(gurcbuild)  gurcbuild.build(['hoge.gura', 'image1.png', 'image2.png']) |

## 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。