

逆引き目次

数値

除算の余りを求めるには %を使う	38 1-6-1	数値を文字列に変換するには number-to-stringを使う	153 5-7-1
累乗を求めるには exptを使う	38 1-6-1	数値を浮動小数点数に変換するには floatを使う	153 5-7-1
1だけ加算するには 1+を使う	38 1-6-1	数値の範囲で場合分けをするには typecaseを使う	205 7-6-2
1だけ減算するには 1-を使う	38 1-6-1	3から10までループするには loop for i from 3 to 10 ...を使う	212 7-9-3
四捨五入を行うには roundを使う	142 5-1-1	1以上 5未満の間繰り返すには loop for i from 1 below 5 ...を使う	213 7-9-4
切捨てを行うには truncateを使う	142 5-1-1	10回繰り返すには loop repeat 10 ...を使う	214 7-9-5
切上げを行うには ceilingを使う	142 5-1-1	合計を求めるには loop ... sum FORMを使う	215 7-9-6
切下げを行うには floorを使う	142 5-1-1	最大値を求めるには loop ... maximize FORMを使う	215 7-9-6
乱数を得るには randomを使う	142 5-1-2	最小値を求めるには loop ... minimize FORMを使う	215 7-9-6

文字列

文字列を結合・オブジェクトの文字列化・書式文字列には formatを使う	42 1-7-4	文字列中の各単語を小文字にそろえるには downcaseを使う	144 5-2-3
文字列を正規表現にマッチさせるには string-matchを使う	71 2-8-6	文字列中の各単語を大文字にそろえるには upcaseを使う	144 5-2-3
正規表現に一致した文字列を得るには match-stringを使う	72 2-8-7	文字列中の各単語をキャピタライズするには capitalizeを使う	144 5-2-3
正規表現を組み立てるには M-x re-builderを使う	73 2-8-8	文字列を正規表現置換するには replace-regexp-in-stringを使う	144 5-2-4
標準出力を文字列にリダイレクトするには with-output-to-stringを使う	97 3-3-2	文字列の長さを得るには lengthを使う	152 5-6-2
文字列を結合するには concatを使う	120 3-8-1	文字列のバイト数を得るには string-bytesを使う	152 5-6-2
文字列をファイルに書き込むには write-regionを使う	124 3-8-5	文字列の幅を得るには string-widthを使う	152 5-6-2
部分文字列を切り出すには substringを使う	143 5-2-1	文字列を数値に変換するには string-to-charを使う	153 5-7-1
文字列をセパレーターで分割するには split-stringを使う	144 5-2-2	文字(数値)を文字列に変換するには char-to-stringを使う	153 5-7-1

文字列を文字(数値)に変換するには <code>string-to-char</code> を使う	153 5-7-1
文字列の集合にマッチする正規表現を作成するには <code>regexp-opt</code> を使う	175 6-7-1
正規表現を読みやすい形式に変換するには <code>rx</code> を使う	176 6-7-2

文字列からS式に変換するには <code>read</code> を使う	177 6-8-1
時刻を人間にわかりやすい文字列にするには <code>format-time-string</code> を使う	298 12-2-1

リスト

コンソセルを作成するには ドット記法か <code>cons</code> を使う	52 2-4-1
コンソセルの左側を取り出すには <code>car</code> を使う	52 2-4-1
コンソセルの右側を取り出すには <code>cdr</code> を使う	52 2-4-1
リストを作成するには リスト記法か <code>list</code> を使う	53 2-4-2
リストの先頭要素を得るには <code>car</code> を使う	53 2-4-2
リストの先頭要素以外を得るには <code>cdr</code> を使う	53 2-4-2
リストの先頭に要素を追加したリストを作成するには <code>cons</code> を使う	53 2-4-2
リストのN番目の要素を得るには <code>nth</code> か <code>elt</code> を使う	53 2-4-2
<code>setq</code> と <code>cons</code> をまとめるには <code>add-to-list</code> を使う	82 2-11-1
値がリストの要素に含まれているかを判定するには <code>memq</code> 、 <code>member</code> を使う	101 3-4-3
同一要素から構成されるリストを作成するには <code>make-list</code> を使う	147 5-4-1
リストにリストを追加するには <code>append</code> を使う	148 5-4-3
リストの要素をひっくり返すには <code>reverse</code> を使う	149 5-4-4
リストをソートするには <code>sort</code> を使う	149 5-4-5
連想リストをソートするには <code>sort</code> を使う	149 5-4-5
リストから要素を削除するには <code>delq</code> や <code>delete</code> を使う	150 5-4-6

リストをコピーするには <code>copy-sequence</code> を使う	152 5-6-1
リストの長さを得るには <code>length</code> を使う	152 5-6-2
連想リストのキーに対応するペアを得るには <code>assq</code> や <code>assoc</code> を使う	159 6-2-2
連想リストの値に対応するペアを得るには <code>rassq</code> や <code>rassoc</code> を使う	159 6-2-3
連想リストのキーに対応する値を得るには <code>assoc-default</code> を使う	159 6-2-4
シーケンスの各要素に順に関数を呼び出し、その結果をリストで得るには <code>mapcar</code> を使う	166 6-5-4
シーケンスの各要素に対し順に関数を呼び出し、その結果をリストで得るには <code>mapconcat</code> を使う	167 6-5-5
リスト構造を分解して各要素を変数に代入するには <code>destructuring-bind</code> を使う	192 7-2-1
リストの先頭に挿入して更新するには <code>push</code> を使う	198 7-3-4
同一要素がないときに、リストの先頭に挿入して更新するには <code>pushnew</code> を使う	198 7-3-4
リスト先頭要素を取り除いて更新するには <code>pop</code> を使う	198 7-3-4
連想リストの各要素をループするには (<code>loop for (K . V) in ALIST ...</code>) を使う	212 7-9-3
リストの各要素に対し順にフォームを評価し、その結果のリストをつなげるには (<code>loop for VAR in LIST append FORM</code>) を使う	215 7-9-6
リストの全要素が条件を満たすか調べるには (<code>loop for VAR in LIST always CONDITION</code>) を使う	215 7-9-6
リストの全要素が条件を満たさないか調べるには (<code>loop for VAR in LIST never CONDITION</code>) を使う	215 7-9-6
リストのどれか要素が条件を満たすか調べるには (<code>loop for VAR in LIST thereis CONDITION</code>) を使う	215 7-9-6

リストの各要素において 条件を満たした回数を数えるには (loop for VAR in LIST count CONDITION)を使う	215 7-9-6	条件を満たす最初の要素を得るには (loop for x in LIST if CONDITION return x)を使う	220 7-9-11
リストから条件を満たす要素を取り出すには (loop for x in LIST if CONDITION collect x)を使う	219 7-9-10		

オブジェクト

代入するには setqを使う	48 2-2-3	同一要素から構成されるベクタを作成するには make-vectorを使う	151 5-5-1
ローカル変数を定義するには letやlet*を使う	50 2-3-1	シーケンスを結合したベクタを作成するには vconcatを使う	151 5-5-2
nilであるかどうか調べるには nullを使う	55 2-4-3	リストをベクタに変換するには vconcatを使う	151 5-5-2
コンソセルであるかどうか調べるには conspを使う	55 2-4-3	ベクタをリストに変換するには appendを使う	153 5-7-1
リストであるかどうか調べるには listpを使う	55 2-4-3	ハッシュテーブルを作成するには make-hash-tableかハッシュテーブル リテラル (Emacs23.2以降) を使う	161 6-3-2
アトム (コンソセル以外) であるか調べるには atomを使う	55 2-4-3	ハッシュテーブルにペアを追加するには puthashを使う	162 6-3-3
シンボルであるかどうか調べるには symbolpを使う	55 2-4-3	ハッシュテーブルから値を取り出すには gethashを使う	162 6-3-3
ベクタを作成するには ベクタ記法か vector を使う	56 2-4-4	ハッシュテーブルから値を削除するには remhashを使う	162 6-3-3
ベクタのN番目の要素を得るには eltか arelを使う	56 2-4-4	ハッシュテーブルのキーのリストを得るには (loop for k being hash-key in ハッシュテーブル collect k)を使う	163 6-3-4
ベクタの要素を設定するには asetを使う	56 2-4-4	ハッシュテーブルの値のリストを得るには (loop for v being hash-values in ハッシュテーブル collect v)を使う	163 6-3-4
2つのオブジェクトを同値比較するには equalを使う	57 2-5-2	ハッシュテーブルから連想リストに変換するには (loop for k being hash-key in ハッシュテーブル using (hash-values v) collect (cons k v))を使う	163 6-3-4
2つのオブジェクトを同一比較するには eqを使う	58 2-5-3	シンボルの属性リストの属性値を得るには getを使う	164 6-4-2
変数を定義 (宣言) するには defvarを使う	81 2-10-3	シンボルの属性リストの属性値を設定するには putを使う	164 6-4-2
シンボル名を得るには symbol-nameを使う	145 5-3-1	変数が定義されているか確かめるには boundpを使う	168 6-5-6
文字列からシンボルを作成するには internを使う	145 5-3-1	汎変数に代入するには setfを使う	195 7-3-1
既存のシンボルがどうか調べるには intern-softを使う	145 5-3-1	ローカルな汎変数を作るには leftfを使う	197 7-3-2
シンボルから変数の値を参照するには symbol-valueを使う	146 5-3-2	構造体を定義するには defstructを使う	209 7-8-2
シンボルからその変数の値を設定するには setを使う	146 5-3-2	変数の別名を定義するには defvaraliasを使う	275 10-5-3

制御構造

真偽値を反転させるには <code>not</code> を使う	59 2-6-1	オブジェクトの型で場合分けするには <code>typecase</code> を使う	205 7-6-2
条件を満たすときに実行するには <code>when</code> を使う	59 2-6-2	レキシカルスコープの非局所脱出するには <code>block</code> と <code>return-from</code> を使う	206 7-7-1
条件を満たさないときに実行するには <code>unless</code> を使う	59 2-6-2	条件を満たす間繰り返すには <code>(loop while CONDITION ...)</code> を使う	214 7-9-5
条件分岐するには <code>if</code> を使う	60 2-6-3	条件を満たさない間繰り返すには <code>(loop until CONDITION ...)</code> を使う	214 7-9-5
複数のフォームを一つのフォームにまとめるには <code>progn</code> を使う	61 2-6-4	<code>loop</code> マクロでループ開始前に設定されるローカル変数を使うには <code>(loop with VAR = FORM ...)</code> を使う	217 7-9-8
複数のフォームをまとめ、1番目の式の値を返すには <code>prog1</code> を使う	61 2-6-4	<code>loop</code> マクロでループのたびに設定されるローカル変数を使うには <code>(loop ... for VAR = FORM ...)</code> を使う	217 7-9-8
複数のフォームをまとめ、2番目の式の値を返すには <code>prog2</code> を使う	61 2-6-4	<code>loop</code> の事前処理をするには <code>(loop initially FORMS ...)</code> を使う	218 7-9-9
条件分岐の処理が複数のフォームにわたすには <code>cond</code> を使う	63 2-6-5	<code>loop</code> の事後処理をするには <code>(loop finally FORMS ...)</code> を使う	218 7-9-9
複数の条件式により分岐するには <code>cond</code> を使う	63 2-6-6	<code>loop</code> の戻り値を指定するには <code>(loop ... finally return FORM)</code> を使う	218 7-9-9
「または」を表現するには <code>or</code> を使う	64 2-6-7	<code>loop</code> で条件分岐するには <code>if</code> 節や <code>unless</code> 節を使う	219 7-9-10
「かつ」を表現するには <code>and</code> を使う	64 2-6-7	<code>loop</code> で複数の計算結果を別の変数に格納するには 演算節に <code>into</code> を使う	219 7-9-10
条件を満たす間ループするには <code>while</code> を使う	66 2-7-1	<code>loop</code> から脱出するには <code>return</code> 節を使う	220 7-9-11
リスト各要素でループするには <code>dolist</code> を使う	67 2-7-2	条件分岐で条件式の値を使うには <code>and</code> を定義して使う	300 12-2-3
式の値で場合分けをするには <code>case</code> を使う	204 7-6-1		

関数

関数を定義するには <code>defun</code> を使う	74 2-9-1	キーボードマクロをコマンド化するには <code>macro-save</code> を定義して使う	81 2-10-3
省略可能引数をつけるには 引数リストで <code>&optional</code> を使う	77 2-9-3	関数や変数のドキュメントや定義を調べるには <code>M-x anything-apropos</code> を使う	158 6-1-4
可変長引数をつけるには 引数リストで <code>&rest</code> を使う	77 2-9-4	関数オブジェクトを呼び出すには <code>funcall</code> を使う	165 6-5-1
無名関数(ラムダ式)を定義するには <code>lambda</code> を使う	78 2-9-5	名前付き関数を関数引数に渡すには シンボルを使う	165 6-5-2
コマンドを定義するには 関数定義の最初に <code>interactive</code> を置く	78 2-10-1	リストの各要素を引数にして関数を呼び出すには <code>apply</code> を使う	166 6-5-3

引数をそのまま返すには <code>identity</code> を使う	167 6-5-5	フックに関数を登録するには <code>add-hook</code> を使う	247 9-3-2
引数をすべて無視して常に <code>nil</code> を返すには <code>ignore</code> を使う	167 6-5-5	フックから関数を削除するには <code>remove-hook</code> を使う	247 9-3-2
関数が定義されているか確かめるには <code>fboundp</code> を使う	168 6-5-6	フックを実行させるには <code>run-hooks</code> を使う	249 9-3-3
複数の関数を同じ引数で順次実行するには <code>run-hook-with-args</code> を使う	169 6-5-7	アドバイスを定義するには <code>defadvice</code> を使う	250 9-4-2
成功するまで関数リストの関数を順次実行するには <code>run-hook-with-args-until-success</code> を使う	169 6-5-8	アドバイスで引数にアクセスするには <code>ad-get-arg</code> や <code>ad-get-args</code> を使う	251 9-4-4
失敗するまで関数リストの関数を順次実行するには <code>run-hook-with-args-until-failure</code> を使う	169 6-5-8	アドバイスで引数を設定するには <code>ad-set-arg</code> や <code>ad-set-args</code> を使う	251 9-4-4
マクロを定義するには <code>defmacro</code> を使う	181 6-9-5	アドバイスで返り値にアクセスするには <code>ad-return-value</code> を読み書きする	252 9-4-5
マクロ内部で使う一意的なシンボルを作成するには <code>(eval-when-compile (require 'cl))</code> 後に <code>gensym</code> を使う	183 6-9-7	<code>around</code> アドバイスで関数本体を呼び出すには 呼び出す部分に <code>ad-do-it</code> を置く	253 9-4-6
マクロの展開結果を得るには <code>macroexpand</code> を使う	195 7-3-1	アドバイスを有効にするには <code>ad-enable-advice</code> の後に <code>ad-activate</code> を使う	254 9-4-8
レキシカルスコープを作成するには <code>lexical-let</code> または <code>lexical-let*</code> を使う	200 7-4-2	アドバイスを無効にするには <code>ad-disable-advice</code> の後に <code>ad-activate</code> を使う	254 9-4-8
一時的に関数を再定義するには <code>flet</code> を使う	201 7-5-1	アドバイス名の正規表現にマッチする すべてのアドバイスを有効にするには <code>ad-enable-regexp</code> の後に <code>ad-activate-regexp</code> を使う	254 9-4-8
ローカル関数を定義するには <code>labels</code> を使う	202 7-5-2	アドバイス名の正規表現にマッチする すべてのアドバイスを無効にするには <code>ad-disable-regexp</code> の後に <code>ad-activate-regexp</code> を使う	254 9-4-8
関数をトレースするには <code>M-x trace-function</code> を使う	231 8-3-2	条件分岐時に関数名が異なり引数と同じ場合には <code>funcall</code> を使って関数名を分岐する	271 10-4-4
正規表現にマッチする関数をすべて得るには <code>(apropos-internal 正規表現 'fboundp)</code> を使う	232 8-3-4	関数の別名を定義するには <code>defalias</code> を使う	274 10-5-2
実行時間を測定するには <code>benchmark-run</code> を使う	239 8-6-1	コマンド終了後に実行する関数を指定するには <code>run-with-timer</code> や <code>run-with-idle-timer</code> で秒数に <code>0</code> に指定する	301 12-2-4

コマンド

<code>interactive</code> をより使いやすくするには 文字列ではなくてフォームを指定する	117 3-6-5	キーマップを作成するには <code>make-keymap</code> か <code>make-sparse-keymap</code> を使う	244 9-1-5
リージョンを引数にするコマンドを作成するには <code>interactive</code> でコード文字 <code>r</code> を指定する	118 3-7-1	キーボード選択できるメニューを使って コマンドを実行するには <code>one-key.el</code> を使う	245 9-2-1
コマンドをキーに割り当てるには <code>define-key</code> を使う	242 9-1-1	キー同時押しでコマンドを実行するには <code>key-chord.el</code> を使う	246 9-2-2
コマンドを <code>global-map</code> に割り当てるには <code>(global-set-key (kbd キー) コマンド)</code> を使う	242 9-1-1	古いコマンドに割り当てられているキーを 新しいコマンドに割り当て直すには <code>define-key</code> に <code>[remap 古いコマンド]</code> を指定する	272 10-4-5
キー割り当てを無効にするには <code>define-key</code> のコマンドに <code>nil</code> を指定する	242 9-1-1		

バッファ

エコーエリアに文字列を表示するには <code>message</code> を使う	78 2-10-1	変数を常にバッファローカル変数にするには <code>make-variable-buffer-local</code> を使う	102 3-4-4
バッファに関連づけられたファイル名を得るには <code>buffer-file-name</code> を参照する	90 3-1-2	モードにローカルな変数を設定するには <code>make-local-variable</code> と <code>set</code> を使う	102 3-4-4
カレントバッファを得るには <code>current-buffer</code> を使う	91 3-1-4	バッファローカル変数のデフォルト値を得るには <code>default-value</code> を使う	102 3-4-4
バッファ名からバッファオブジェクトを得るには <code>get-buffer</code> を使う	92 3-1-5	バッファローカル変数のデフォルト値を設定するには <code>setq-default</code> を使う	102 3-4-4
全バッファオブジェクトのリストを得るには <code>buffer-list</code> を使う	92 3-1-5	現在位置を得るには <code>point</code> を使う	103 3-4-5
カレントバッファを切り換え、 現在のウィンドウに表示するには <code>switch-to-buffer</code> を使う	92 3-1-6	バッファ先頭での位置を得るには <code>point-min</code> を使う	103 3-4-5
バッファが存在しないときに新規作成するには <code>get-buffer-create</code> を使う	93 3-2-1	バッファ末尾での位置を得るには <code>point-max</code> を使う	103 3-4-5
バッファを新規作成するには <code>generate-new-buffer</code> を使う	93 3-2-2	行頭の位置を得るには <code>point-at-bol</code> を使う	103 3-4-5
ファイルをバッファに読み込み、表示するには <code>find-file</code> を使う	94 3-2-3	行末の位置を得るには <code>point-at-eol</code> を使う	103 3-4-5
ファイルをバッファに読み込むには <code>find-file-noselect</code> を使う	94 3-2-3	バッファ先頭であるか調べるには <code>bobp</code> を使う	103 3-4-5
バッファを削除するには <code>kill-buffer</code> を使う	94 3-2-4	バッファ末尾であるか調べるには <code>eobp</code> を使う	103 3-4-5
バッファオブジェクトの生存確認するには <code>buffer-live-p</code> を使う	94 3-2-4	行頭であるか調べるには <code>bolp</code> を使う	103 3-4-5
バッファ名から生存確認をするには <code>get-buffer</code> を使う	94 3-2-4	行末であるか調べるには <code>eolp</code> を使う	103 3-4-5
標準出力に出力するには <code>princ</code> を使う	97 3-3-1	ポイント直前の文字を得るには <code>char-before</code> を使う	103 3-4-5
標準出力をバッファにリダイレクトし、表示するには <code>with-output-to-temp-buffer</code> を使う	98 3-3-3	ポイント直後の文字を得るには <code>char-after</code> を使う	103 3-4-5
カレントバッファを切り換えながら フォームを評価するには <code>with-current-buffer</code> を使う	99 3-4-1	バッファに文字列を書き込むには <code>insert</code> を使う	105 3-5-1
一時バッファ内でフォームを評価するには <code>with-temp-buffer</code> を使う	99 3-4-1	他のバッファの内容(の一部)を書き込むには <code>insert-buffer-substring</code> を使う	105 3-5-1
バッファ名を得るには <code>buffer-name</code> を使う	99 3-4-2	バッファの内容を文字列で得るには <code>buffer-string</code> を使う	105 3-5-1
バッファ名を変更するには <code>rename-buffer</code> を使う	99 3-4-2	バッファの内容の一部を文字列で得るには <code>buffer-substring</code> を使う	105 3-5-1
バッファ名を変更する際に、 自動でバッファ名を決定するには <code>rename-uniquely</code> を使う	99 3-4-2	カーソルを任意の場所に移動するには <code>goto-char</code> を使う	106 3-5-2
現在のバッファのメジャーモードを知るには <code>major-mode</code> にアクセスする	101 3-4-3	バッファ先頭へ移動には <code>(goto-char (point-min))</code> と記述する	106 3-5-2

バッファ末尾へ移動するには (goto-char (point-max))と記述する	106 3-5-2	後方へ正規表現検索を行うには re-search-backward を使う	109 3-5-5
行頭へ移動するには beginning-of-line を使う	106 3-5-2	正規表現検索後にマッチした文字列を得るには match-string を使う	109 3-5-5
行末へ移動するには end-of-line を使う	106 3-5-2	キルリングにアクセスするには kill-ring を参照する	111 3-5-7
N文字分進むには forward-char を使う	106 3-5-2	キルリングに新たなテキストを格納するには kill-new を使う	111 3-5-7
N行分進むには forward-line を使う	106 3-5-2	編集領域を限定(ナローイング)するには narrow-to-region を使う	112 3-5-8
N単語分進むには forward-word を使う	106 3-5-2	ナローイングを解除するには widen を使う	112 3-5-8
N個のS式分進むには forward-sexp を使う	106 3-5-2	ポイントをマーカーに変換するには (set-marker (make-marker 位置))を使う	119 3-7-2
Nシンボル分進むには forward-symbol を使う	106 3-5-2	既存のマーカーの位置を再設定するには move-marker を使う	119 3-7-2
Nページ分進むには forward-page を使う	106 3-5-2	マーカーが指しているバッファを得るには marker-buffer を使う	119 3-7-2
行番号を指定してカーソル移動するには goto-line を使う	106 3-5-2	マーカーがどこかを指しているかを調べるには marker-position を使う	119 3-7-2
バッファの内容をクリアするには erase-buffer を使う	108 3-5-3	バッファにファイル内容を挿入するには insert-file-contents を使う	124 3-8-4
リージョンを削除するには delete-region を使う	108 3-5-3	ファイルバッファをファイルに保存するには save-buffer を使う	124 3-8-5
カーソル直後の文字を削除するには delete-char を使う	108 3-5-3	任意のバッファ(一部分でも)を ファイルに保存するには write-region を使う	124 3-8-5
カーソル直前の文字を削除するには delete-backward-char を使う	108 3-5-3	バッファをファイルに書き込むときに エンコーディングを指定するには coding-system-for-write を指定する	124 3-8-5
カレントバッファ・ポイント・マーカーを保存して、108 フォーム呼び出し後に復元するには save-excursion を使う	108 3-5-4	バッファからS式を読み込むには read を使う	177 6-8-1
前方へ文字列検索を行うには search-forward を使う	109 3-5-5	明示的に font-lock を反映させるには font-lock-fontify-buffer を使う	271 10-4-4
後方へ文字列検索を行うには search-backward を使う	109 3-5-5	カーソル位置直後の文字列が 正規表現を満たすかどうかを調べるには looking-at を使う	324 13-4-4
前方へ正規表現検索を行うには re-search-forward を使う	109 3-5-5		

ファイル

Lispファイルを置くディレクトリを指定するには load-path で指定する	82 2-11-1	Lispファイルを1度だけロードするには require を使う	83 2-11-3
Lispファイルをロードするには load を使う	83 2-11-2	require を取り消すには unload-feature を使う	84 2-11-3

パス名からディレクトリを抜き出すには <code>file-name-directory</code> を使う	120 3-8-1
パス名からファイル名を抜き出すには <code>file-name-nondirectory</code> を使う	120 3-8-1
パス名から拡張子を抜き出すには <code>file-name-extension</code> を使う	120 3-8-1
パス名から拡張子以外を抜き出すには <code>file-name-sans-extension</code> を使う	120 3-8-1
ディレクトリに「/」を付けるには <code>file-name-as-directory</code> を使う	120 3-8-1
ディレクトリから「/」を外すには <code>directory-file-name</code> を使う	120 3-8-1
相対パスを得るには <code>file-relative-name</code> を使う	120 3-8-1
ディレクトリ以下のファイルのリストを得るには <code>directory-files</code> を使う	122 3-8-2
ワイルドカードを展開するには (<code>require 'em-glob</code>)の後で <code>eshell-extended-glob</code> を使う	123 3-8-2

ファイルが存在するか確かめるには <code>file-exists-p</code> を使う	123 3-8-3
ファイルに読み込み権限があるか確かめるには <code>file-readable-p</code> を使う	123 3-8-3
ファイルに書き込み権限があるか確かめるには <code>file-writable-p</code> を使う	123 3-8-3
ファイルに実行権限があるか確かめるには <code>file-executable-p</code> を使う	123 3-8-3
シンボリックリンクであるか確かめるには <code>file-symlink-p</code> を使う	123 3-8-3
ディレクトリであるか確かめるには <code>file-directory-p</code> を使う	123 3-8-3
通常のファイルであるか確かめるには <code>file-regular-p</code> を使う	123 3-8-3
絶対パスであるか確かめるには <code>file-name-absolute-p</code> を使う	123 3-8-3
オブジェクトを永続化するには <code>marshal-dump</code> と <code>marshal-load</code> を定義して使う	178 6-8-2

ミニバッファ

ミニバッファから文字列を読み込むには <code>read-string</code> を使う	113 3-6-1
補完つきで文字列を読み込むには <code>completing-read</code> を使う	114 3-6-2
「はい」か「いいえ」かをユーザに尋ねるには <code>y-or-n-p</code> を使う	115 3-6-3
ファイル名を読み込むには <code>read-file-name</code> を使う	116 3-6-4
バッファ名を読み込むには <code>read-buffer</code> を使う	116 3-6-4

数値を読み込むには <code>read-number</code> を使う	116 3-6-4
パスワードを読み込むには <code>read-passwd</code> を使う	116 3-6-4
コマンド名を読み込むには <code>read-command</code> を使う	116 3-6-4
変数名を読み込むには <code>read-variable</code> を使う	116 3-6-4

ウィンドウ

選択されたウィンドウを得るには <code>selected-window</code> を使う	128 4-1-1
ウィンドウ上下分割するには <code>split-window-vertically</code> を使う	128 4-1-2
ウィンドウ左右分割するには <code>split-window-horizontally</code> を使う	128 4-1-2
ウィンドウを削除するには <code>delete-window</code> を使う	130 4-2-1
特定のウィンドウ以外を削除するには <code>delete-other-windows</code> を使う	130 4-2-2

ウィンドウを選択するには <code>select-window</code> を使う	131 4-3-1
選択されたウィンドウを保存して、 フォーム呼び出し後に復元するには <code>save-selected-window</code> を使う	131 4-3-1
他のウィンドウを選択するには <code>other-window</code> を使う	132 4-3-2
ウィンドウが表示しているバッファを得るには <code>window-buffer</code> を使う	132 4-4-1
バッファを表示しているウィンドウを得るには <code>get-buffer-window</code> を使う	132 4-4-2

選択されたウィンドウに表示している バッファを切り換えるには <code>switch-to-buffer</code> を使う	133 4-5-1	左右分割されたウィンドウのサイズを広げるには <code>enlarge-window-horizontally</code> を使う	137 4-7-2
別ウィンドウを選択してバッファを表示するには <code>pop-to-buffer</code> を使う	133 4-5-1	上下分割されたウィンドウのサイズを縮めるには <code>shrink-window</code> を使う	137 4-7-2
ユーザに読ませる目的でバッファを表示するには <code>display-buffer</code> を使う	133 4-5-2	左右分割されたウィンドウのサイズを縮めるには <code>shrink-window-horizontally</code> を使う	137 4-7-2
選択されたウィンドウを一時的に切り換えるには <code>with-selected-window</code> を使う	135 4-6-1	バッファの内容がびったり収まるように ウィンドウを縮小するには <code>shrink-window-if-larger-than-buffer</code> を使う	138 4-7-3
ウィンドウの高さを得るには <code>window-height</code> を使う	137 4-7-1	現在のウィンドウ構成を得るには <code>current-window-configuration</code> を使う	139 4-8-1
ウィンドウの幅を得るには <code>window-width</code> を使う	137 4-7-1	ウィンドウ構成を復元するには <code>set-window-configuration</code> を使う	139 4-8-1
上下分割されたウィンドウのサイズを広げるには <code>enlarge-window</code> を使う	137 4-7-2	ウィンドウ構成を保存して、 フォーム呼び出し後に復元するには <code>save-window-excursion</code> を使う	140 4-8-2

プロセス

シェルコマンドを実行するには <code>shell-command</code> を使う	278 11-1-2	終了時に処理を行わせるには <code>set-process-sentinel</code> を使う	283 11-3-3
シェルコマンドの実行結果を文字列で得るには <code>shell-command-to-string</code> を使う	278 11-1-2	プロセスが動作しているバッファを得るには <code>process-buffer</code> を使う	283 11-3-3
リージョンを入力としてシェルコマンドを実行するには <code>shell-command-on-region</code> を使う	278 11-1-2	プロセスに文字列を送信するには <code>process-send-string</code> を使う	284 11-3-4
シェルのメタ文字をエスケープするには <code>shell-quote-argument</code> を使う	279 11-1-2	プロセスの評価結果の文字列の 遅延オブジェクトを作成するには <code>deferred:process</code> を使う	303 12-3-5
環境変数の値を得るには <code>getenv</code> を使う	280 11-2-1	プロセスバッファの遅延オブジェクトを作成するには <code>deferred:process-buffer</code> を使う	303 12-3-5
環境変数の値を設定するには <code>setenv</code> を使う	280 11-2-1	URLの内容を取得するには <code>deferred:url-retrieve</code> を使う	305 12-3-7
外部プログラムの存在確認をし、 存在する場合にフルパスを得るには <code>executable-find</code> を使う	281 11-2-2	シェルコマンドの実行結果を挿入には <code>call-process-shell-command</code> を使う	316 13-3-5
非同期プロセスを作成するには <code>start-process</code> や <code>start-process-shell-command</code> を使う	282 11-3-2		

anything.el

ファイルやバッファに関する anythingコマンドを利用するには <code>M-x anything-for-files</code> や <code>M-x anything-filelist+</code> を使う	309 13-1-3	anythingコマンドを作成するには <code>anything-other-buffer</code> を 呼び出すコマンドを定義する	313 13-2-2
anythingコマンドを選択して実行するには <code>M-x anything-execute-anything-command</code> を使う	311 13-1-5	情報源で情報源の名前を指定するには <code>name</code> 属性を使う	314 13-3-1
情報源を選択して実行するには <code>M-x anything-call-source</code> を使う	312 13-2-1	情報源で候補関数を指定するには <code>candidates</code> 属性を使う	315 13-3-2

情報源でアクションを指定するには <code>action</code> 属性を使う	315 13-3-3
情報源でバッファの各行を候補するには <code>candidates-in-buffer</code> 、 <code>init</code> 属性を使う	316 13-3-5
情報源で候補バッファを作成するには <code>anything-candidate-buffer</code> を使う	316 13-3-5
情報源で候補ファイルを指定するには <code>candidates-file</code> 属性を使う	318 13-3-6
情報源で候補の型を指定するには <code>type</code> 属性を使う	318 13-3-7
情報源をMigemo対応にするには <code>(migemo)</code> を加える	319 13-3-8
情報源で候補ファイルをgrep検索するには <code>grep-candidates</code> 属性を使う	321 13-4-2
情報源で表示候補数を制限するには <code>candidate-number-limit</code> 属性を使う	321 13-4-2
一定文字数を入力して初めて起動する 情報源を作成するには <code>requires-pattern</code> 属性を使う	321 13-4-2

型を定義するには <code>define-anything-type-attribute</code> を使う	322 13-4-3
情報源でC-zを押したときに実行するアクション (<code>persistent-action</code>)を指定するには <code>persistent-action</code> 属性を使う	322 13-4-3
情報源でpersistent-actionの説明をするには <code>persistent-help</code> 属性を使う	322 13-4-3
情報源でアクションに渡るオブジェクトから 表示文字列へ変換するには <code>real-to-display</code> 属性を使う	322 13-4-3
情報源で表示文字列からアクションに渡る オブジェクトに変換するには <code>display-to-real</code> 属性を使う	322 13-4-3
persistent-actionと普通のactionを区別するには <code>anything-in-persistent-action</code> を調べる	324 13-4-4
anything/バッファの現在行を書き換えるには <code>anything-edit-current-selection</code> を使う	324 13-4-4
型の定義を変更するには <code>anything-c-arrange-type-attribute</code> を使う	327 13-5-2
プラグインを定義するには <code>anything-compile-source-functions</code> に プラグイン展開関数を追加する	328 13-5-4

その他

エラーメッセージを表示して <code>elisp</code> プログラムの実行を終了するには <code>error</code> を使う	171 6-6-2
エラーシグナルを指定してエラーを通知するには <code>signal</code> を使う	171 6-6-3
エラーを無視してnilを返すようにするには <code>ignore-errors</code> を使う	172 6-6-4
エラーを捕捉するには <code>condition-case</code> を使う	172 6-6-5
エラーメッセージを文字列で得るには <code>error-message-string</code> を使う	172 6-6-5
エラーが起きても必ず実行される 後片付け処理を記述するには <code>unwind-protect</code> を使う	174 6-6-6
S式を評価するには <code>eval</code> を使う	177 6-8-1
Emacsのバージョンを得るには <code>emacs-major-version</code> 、 <code>emacs-major-version</code> 、 <code>emacs-minor-version</code> を参照する	185 6-10-1
プラットフォームを得るには 変数 <code>system-type</code> にアクセスする	186 6-10-3

メジャーモードを作成するには <code>define-derived-mode</code> を使う	258 10-1-1
ファイル名とメジャーモードを関連付けるには <code>auto-mode-alist</code> を設定する	259 10-1-3
コメント文字・キーワード・ファイルの正規表現を 指定してメジャーモードを作成するには <code>define-generic-mode</code> を使う	261 10-2-3
font-lockの設定を追加するには <code>font-lock-add-keywords</code> を使う	267 10-3-4
マイナーモードを定義するには <code>define-minor-mode</code> を使う	269 10-4-3
Emacsの動作をN秒一時停止するには <code>sit-for</code> や <code>sleep-for</code> を使う	283 11-3-3
未来(特定の時刻・現在からの経過時間) に関数を呼び出すには <code>run-with-timer</code> あるいは <code>run-at-time</code> を使う	298 12-2-1
タイマーを無効にするには <code>cancel-timer</code> を使う	298 12-2-1
アイドルタイマーを設置するには <code>run-with-idle-timer</code> を使う	299 12-2-2