

## 内容

1. 関数マニュアル .....	2
1.1. 文字関数 .....	2
1.2. 数値演算 .....	7
1.3. ファイル操作関数.....	10
1.4. データベース関数 .....	13
1.5. 配列関数 .....	16
1.6. JSON 関連.....	16
1.7. その他 .....	18

## 関数一覧

### 1. 関数マニュアル

ここでは Lutino JavaScript の関数一覧を表示します。基本的に Lutino では関数を利用してプログラムをすることになります。そのため様々な関数が用意してあります。

関数マニュアルはある程度プログラムを書ける人を対象としています。分からない文言などは AI に聞いてみたり、問い合わせをしてみてください。

#### 1.1. 文字関数

Lutino 内部では文字コードとして utf-8 を想定しています。

しかしながら便利のため windows 版でファイル名だけは SJIS で扱えるようにしています。ファイルの中身は変換されません。例えば CSV ファイルは通常 SJIS です。これを読むためにはコード変換が必要です。(Lutino の SQL Browser ではこの問題に対応してあります。)

- `charToInt(string ch)`

概要:

文字を整数に変換する

戻り値:

文字に変換した数値

例:

```
var number = charToInt("1");  
// Expected output: 49
```

- `String.indexOf(string search)`

概要:

文字列の出現位置を返す

戻り値:

指定した文字列の出現位置、見つからなければ-1

例:

```
var str = "abcdefg";  
var position = str.indexOf("bcd");  
// Expected output: 1
```

- `String.substring(int lo,int hi)`

概要:

string の lo+1 番目から hi+1 番目の部分文字列を返す

戻り値:

部分文字列

例:

```
var str = "abcdefg";  
var str2 = str.substring(1,3);  
// Expected output: "bc"
```

- String.charAt(int pos)

概要:

文字列の pos 位置の 1 文字を返す。全角文字非対応

戻り値:

切り出したアスキー文字1文字分

例:

```
var str = "abcdefg";  
var ch = str.charAt(2);  
// Expected output: "c"
```

- String.charCodeAt(int pos)

概要:

文字列の pos 位置の文字コードを返す。全角文字非対応

戻り値:

切り出したアスキー文字1文字分の文字コード

例:

```
var str = "abcdefg";  
var code = str.charCodeAt (2);  
// Expected output: 98
```

- String.fromCharCode(int code)

概要:

指定された文字コードに対応する文字を返す。全角文字非対応

戻り値:

code で指定された文字

例:

```
var char = str.fromCharCode (98);  
// Expected output: "b"
```

## 関数一覧

- String.split(string separator)

概要:

separator で指定された文字で文字列を分解し配列として返す

結合には Array.join を使う

戻り値:

文字の配列

例:

```
var str = "a,b,c,d,e,f,g";  
var ary = str. split(",");  
// Expected output: ["a","b","c","d","e","f","g"]
```

- String.replace(string before,string after)

概要:

String で指定される文字列の最初に出現した before の文字を after に置換する

戻り値:

置換した文字列 (String の文字列は変更されない)

例:

```
var str = "abcdefg";  
var replace = str. replace("abc","xyz");  
// Expected output: "xyzdefg"
```

- String.preg\_replace(string pattern, string replace)

概要:

戻り値:

pattern で指定された正規表現文字列を replace に置換する。

pattern は配列指定可

戻り値: 置換した文字列

String.addSlashes()

概要:

戻り値:

文字列中の特殊文字(" ' \)をエスケープする

戻り値: エスケープした文字列

例:

```
var str = "a\"'\b\c\d\\efg";
```

## 関数一覧

```
var replace = str. addSlashes();  
// Expected output: " a\"'\bcd\\efg"
```

String.toLowerCase()

概要:

戻り値:

文字列中のアスキー文字を小文字に変換する

戻り値: 変換した文字列

例:

```
var str = "ABCDEFGH";  
var replace = str. toLowerCase ();  
// Expected output: " abcdefg "
```

String.toUpperCase()

概要:

文字列中のアスキー文字を大文字に変換する

戻り値:

変換した文字列

例:

```
var str = "abcdefg";  
var replace = str. toUpperCase ();  
// Expected output: " ABCDEFG "
```

- String.nkfconv(string Format)

概要:

String を Format で示される文字コードに変換します。

入力文字列

省略時: 自動判定

W:utf-8

S:SJIS

E:EUC

出力文字列

省略時: SJIS

w:utf-8

s:SJIS

e:EUC

戻り値:

変換した文字列

例:

下のコードは文字列 title を utf-8 から SJIS に変換した値を返す

```
title.nkfconv("Ws");
```

- `btoa(string data)`

概要:

バイナリデータを base64 文字列に変換する。通常の JavaScript では `btoa`, `atob` は正しく動作しないので非推奨であるが、Lutino では正しく動作する。

戻り値:

変換した文字列

例:

```
var str = "YWJjZGVmZw==";  
var replace = btoa(str);  
// Expected output: "abcdefg"
```

- `atob(string data)`

概要:

base64 文字列をバイナリデータに変換する。通常の JavaScript では `btoa`, `atob` は正しく動作しないので非推奨であるが、Lutino では正しく動作する。

戻り値:

変換したバイナリ文字列

例:

```
var str = "abcdefg";  
var replace = atob(str);  
// Expected output: "YWJjZGVmZw=="
```

- `String.startsWith(string pattern)`

概要:

文字列が `pattern` から始まっているならば `true` そうでなければ `false` を返す

戻り値:

真偽値

## 関数一覧

例:

```
var str = "abcdefg";  
var isStart = str.startsWith("abc");  
// Expected output: true
```

- String.endsWith(string pattern)

概要:

文字列が pattern で終わっていれば true そうでなければ false を返す

戻り値:

真偽値

例:

```
var str = "abcdefg";  
var isEnd = str.endsWith("efg");  
// Expected output: true
```

### 1.2. 数値演算

- Math.rand()

概要:

乱数を返す

戻り値:

0x7fff までの乱数

例:

```
var num = Math.rand();  
// Expected output: xxx
```

- Math.randint(min, max)

概要:

min から max の間の整数の乱数を返す(0x7fff 以下)

戻り値:

min から max の間の乱数

例:

```
var num = Math.randint(10, 20);  
// Expected output: 10~20
```

Integer.parseInt(str)

- 文字から数値への変換

概要:

文字列から整数への変換

## 関数一覧

戻り値:

変換された数値

例:

```
var num = Integer.parseInt("12");  
// Expected output: 12
```

- Integer.valueOf(str)

概要:

1文字を数値に変換

戻り値:

先頭1文字のアスキーコード

例:

```
var num = Integer.valueOf("1");  
// Expected output: 49
```



## 関数一覧

通常関数は以下に一覧表示します。

関数名	説明
Math.Abs(a)	絶対値
Math.round(a)	値の丸め
Math.min(a,b)	最小値
Math.max(a,b)	最大値
Math.range(x,a,b)	
Math.sign(a)	符号
Math.PI()	円周率
Math.toDegrees(a)	度に変換
Math.toRadians(a)	ラジアンに変換
Math.sin(a)	正弦
Math.asin(a)	逆正弦
Math.cos(a)	余弦
Math.acos(a)	逆余弦
Math.tan(a)	正接
Math.atan(a)	逆正接
Math.sinh(a)	双曲線正弦
Math.asinh(a)	逆双極性制限
Math.cosh(a)	双曲線余弦
Math.acosh(a)	逆双曲線余弦
Math.tanh(a)	双曲線正接
Math.atanh(a)	逆双曲線正接
Math.E()	自然対数の底
Math.log(a)	対数
Math.log10(a)	10の対数
Math.exp(a)	自然対数
Math.pow(a,b)	指数
Math.sqr(a)	自乗
Math.sqrt(a)	平方根

## 関数一覧

### 1.3. ファイル操作関数

パスの区切りは windows では\、Linux では/に変換されます。

ただし windows でのドライブ名は c:/users/name/のように表記してください。

windows では相対パスは動作しません。必ずフルパス(C://からすべてのディレクトリの記述)で記述してください。

Lutino が動作するドキュメントルートは\_SERVER.DOCUMENT\_ROOT に格納されています。この値を使ってパスを作成してください。

- `file_exists(string path)`  
ファイルの存在チェック。パスはフルパスで utf-8 とします。  
戻り値: 成功:true 失敗:false  

```
var path="c:\test.text";  
var ret = file_exists(path);
```
- `dir_exists (string path)`  
ディレクトリの存在チェック  

```
var path="/sample";  
var ret = dir_exists (path);
```
- `dirname(string url)`  
url にパスとファイルが含まれるとき、パス名を文字列として返す  

```
var root  = _GET.root;  
var filePath = dirname(root);
```
- `basename(string url)`  
url のファイル名のみを文字列として返す  

```
var url  = _GET.DOCUMENT_ROOT;  
var name = basename(url);
```
- `scandir(string url)`  
url のフォルダ情報を json 文字列として返す  

```
var url  = _GET.DOCUMENT_ROOT;  
files = eval(scandir(url));
```
- `extractFileExt(string url)`  
url の拡張子情報を文字列として返す

## 関数一覧

```
var path = "c:\temp\test.html";  
var ext = extractFileExt(path);
```

- file\_stat(string path)  
ファイルの情報を JSON 形式で取得する。  

```
var path = "c:\temp\test.html";  
var info = file_stat(path);
```
- filedate(string path)  
ファイルの日付情報を文字列として取得する。  

```
var path = "c:\temp\test.data";  
stat = eval(file_stat(path));
```
- loadFromFile(string path)  
path で示されるファイルを文字列として読み込む。  
バイナリファイルを読み込む事ができる。  
path には URL を指定できる。https は未サポート  
// ファイル読みこみ  

```
var data_file = loadFromFile("c:\temp\test.csv");  
※ファイル読み込み時には環境変数が使えます。  
// URL 読み込み  
var data_url = loadFromFile("http://google.co.jp");
```
- loadFromCSV(string path)  
path で示される CSV ファイルを JSON 形式に変換する  

```
var csv = loadFromFile("c:\temp\test.csv");
```
- unlink(string path)  
path で示されるファイルを削除する  

```
unlink("c:\temp\nouse.txt");
```
- touch(string path)  
path で示されるファイルを更新する。  
ファイルが存在しない場合にはファイルを作成する  

```
touch("c:\temp\blank.txt");
```

## 関数一覧

- `rename(string pathf,string patht)`  
ファイルを pathf から patht へリネームする
- `mkdir(string path)`  
path で示されるディレクトリを作成する  
多段階のディレクトリは作成できない。  
ファイルの末尾は"/"の有無は自動設定する
- `rmdir(string path)`  
path で示されるディレクトリを削除する  
中身が空でないディレクトリは削除できない
- `saveToFile(string path,string data)`  
path で示されるファイルに data を保存する
- `copy(pathFrom,pathTo)`  
pathFrom から pathTo へファイルをコピーする

### 1.4. データベース関数

データベースは常時使用可能です。

複数のリクエストが来た時にカウントアップされて、インスタンスがメモリに残ります。たくさんアクセスがあれば自動的に早くなります。

- DBConnect(string DBName)

概要:

DBName で示されるデータベースに接続する。

戻り値:

接続できたなら、接続文字列が返される。

- String.DBDisconnect()

概要:

データベースを切断する。接続文字列を含む文字型が必要

戻り値:

なし。

- String.SQL(string sqltext)

概要:

接続されたデータベースに sqlText で示される SQL を発行する。接続文字列を含む文字型が必要

戻り値:

JSON 形式の結果。エラーの場合は文字列のオブジェクトが出力される。

結果は JSON 形式で取得する。セキュリティ的に保護するものは現在用意されてないので SQL インジェクション等には留意すること

例:

```
var connectString=DBConnect("lesson");
var result = connectString.SQL("select * from student;");
connectString.DBDisconnect();
result.dump();
```

- データベース使用(コマンドラインのみ)

USE データベース名;

指定のデータベースを使用する。

## 関数一覧

- SQL 一覧

カラム名は```,`"で囲う

テーブル名は```,`"などで囲う

文字列は```,`"で囲う

条件式は [[テーブル名./テーブルエイリアス.]カラム名/数値/文字]

[=/>/</>=<=] [[テーブル名./テーブルエイリアス.]カラム名/数値/文字]

## 関数一覧

動作	SQL
選択	SELECT [[テーブル名.]カラム名..[AS ALIAS] * count(*)] FROM table_name1[ AS TABLE_ALILAS1][,table_name2.. [where 条件式 1 [[AND/OR] 条件式 2.. [ORDER BY カラム名 [ASC/DESC]] [LIMIT 行番号 1[,行番号2]]
挿入	INSERT INTO テーブル名(カラム名 1,...) VALUES(値 1,...); ※カラム名のリストはテーブルカラム一覧と合致しなければならない。
更新	UPDATE テーブル名 SET カラム名 1 = 値 1,[カラム名 2=値 2].. WHERE 条 件式 1 [[AND/OR] 条件式 2]...
削除	DELETE FROM テーブル名 WHERE 条件式 1 [[AND/OR] 条件式 2]...
DB作成	CREATE DATABASE データベース名; データベースを作成する。データベース名は他のデータベース名と同じ ではない文字列
DB削除	DROP DATABASE データベース名; 現在使用していないデータベースを抹消する。 データはデータベース名.db のファイル名として残る。 次回同名のデータベースを作成すると復活する
テーブル 作成	CREATE TABLE テーブル名(カラム名 1 number/string, カラム名 2 number/string..); テーブルを現在使用中のデータベースに作成する。起動時には _SYSTEM というデータベースを使用している。 よって何かしらのユーザー用データベースは前もって作成しておくこと。 sqlBrowser にて対話的に作成可能
	CREATE TABLE テーブル名 FROM CSVFILE 指定の CSVFILE からテーブルを作成し全データを INSERT する。カラム は基本文字型となる。カラム名の後に:string を付け加えると文字 型、:number を付け加えると数値型となる。
テーブル 削除	DROP TABLE テーブル名; 使用中のデータベースからテーブルを抹消する。内部のデータも消える
DB一覧	SHOW DATABASES; 現存するデータベース一覧を表示する
テーブル 一覧	SHOW TABLES; 使用中のデータベースのテーブル一覧を表示する
テーブル 変更	ALTER TABLE テーブル名 ADD(カラム名 1 number/string,[...]...) 指定のテーブルに指定のカラムを追加する
	ALTER TABLE テーブル名 MODIFY(カラム名 1 number/string[...]) 指定のテーブルの指定のカラムについて属性を変更する
	ALTER TABLE テーブル名 DROP(カラム名 1,...) 指定のテーブルの指定のカラムについて属性を変更する

## 関数一覧

### 1.5. 配列関数

配列の長さ等操作する関数です。

- `Array.length`

概要:

配列の要素の個数を数えます

戻り値:

配列以外は `undefined`、配列の場合は配列の個数、空の配列は0を返します。

- `Array.contains(object obj)`

概要:

配列が `obj` で指定された値を含むかどうかを調べます。

戻り値:

配列が `obj` を含むならば `true`、そうでない場合は `false` を返します。

- `Array.remove(object obj)`

概要:

配列から `obj` を含む項目を削除する

戻り値:

- `Array.join(string separator)`

概要:

配列を `separator` でつないで文字列にする

戻り値:

結合した文字列を返します。

### 1.6. JSON 関連

- `JSON.mp3id3tag(string path)`

概要:

`path` で示される mp3 ファイルの情報を JSON 形式の文字列で返します。

戻り値:

ファイルが mp3 ではないか無効の場合は項目が空の JSON 形式のデータが返ります。



## 関数一覧

例:

```
var mp3data = eval(JSON.mp3id3tag("mp3file.mp3"));
```

- JSON.stringify(object obj, string replacer)

概要:

obj で示されるオブジェクトを JSON 形式の文字列に変換し返します。

replacer は現状では無効です。""を指定してください。

戻り値:

JSON 文字列が返ります。

例:

```
var a={"a":1,"b":2,"c":3};  
var b=JSON.stringify(a,"");
```

- mimeTypeInfo(string path)

概要:

```
var mime = eval(mimeTypeInfo(path));
```

mimeType.fileType によりサーバー上でどのように扱われるかの mimeType を取得する。

戻り値:

オブジェクト{"mimeType":文字列,"fileType":ファイルタイプ}

```
{"mimeType":"text/markdown","fileType":"TYPE_DOCUMENT"}
```

他に String.SQL、LoadFromCSV もJSONを返します

### 1.7. その他

- `print(string text)`  
ブラウザへのレスポンスに含まれるメッセージボディにテキストを表示します。この表示を行った後では `header` は変更できません。
- `String.toDateString(string format)`  
ファイル日付を、時間を示す文字列に変換する。
  - %a - 現在のロケールに基づく短縮された曜日の名前
  - %A - 現在のロケールに基づく完全な曜日の名前
  - %b - 現在のロケールに基づく短縮された月の名前
  - %B - 現在のロケールに基づく完全な月の名前
  - %c - 現在のロケールに基づく適当な日付と時間の表現
  - %C - 世紀 (年を 100 で割り、整数に丸めたもの。00 から 99)
  - %d - 日付を 10 進数で (01 から 31)。
  - %D - %m/%d/%y と同じ
  - %e - 月単位の日付を 10 進数で表したものの。日付が 1 桁の場合は、前に空白を一つ付けます ('1' から '31')。
  - %g - 世紀以外は %G と同じ。
  - %G - ISO 週番号 (%V を参照) に対応する 4 桁の年。これは ISO 週番号が前年もしくは次年に属するかによって使用される年が異なる事を除き %Y と同じフォーマットと値です。
  - %h - %b と同じ。
  - %H - 時間を 24 時間表示の 10 進数で (00 から 23 まで)。
  - %I - 時間を 12 時間表示の 10 進数で (01 から 12 まで)。
  - %j - 年間での日付を 10 進数で表現 (001 から 366)。
  - %m - 月を 10 進数で表現 (01 から 12)。
  - %M - 分を 10 進数で表現。
  - %n - 改行文字。
  - %p - 指定した時間により 'am' または 'pm'、または現在のロケールの、それに対応する文字列。
  - %r - a.m. および p.m. 表記で表した時間。
  - %R - 24 時間表記で表した時間。
  - %S - 秒を 10 進数で表現。
  - %t - タブ文字。
  - %T - 現在の時間。 %H:%M:%S に等しい。
  - %u - 10 進数表記の曜日。1 から 7 の範囲で表し、1 が月曜日。

## 関数一覧

%U - 年間で何番目の週であるかを 10 進数で表現。年間で最初の日曜を最初の

週の最初の日として数えます。

%V - ISO 8601:1988 で規定された現在の年の週番号の 10 進数表現で、01 から 53 までの範囲となります。1 は最初の週で、その週は現在の年に最低 4 日はあります。週は月曜日から始まります (指定したタイムスタンプの 週番号に対応する年を表すには、%G あるいは %g をしてください)。

%W - 現在の年で何番目の週であるかを 10 進数で表現。年間で最初の月曜を

最初の週の最初の日として数えます。

%w - 曜日を 10 進数で表現。日曜は 0 になります。

%x - 時間を除いた日付を現在のロケールに基づき表現します。

%X - 日付を除いた時間を現在のロケールに基づき表現します。

%y - 世紀の部分を除いた年を 10 進数として表現 (00 から 99 までの範囲)。

%Y - 世紀を含む年を 10 進数で表現。

%Z あるいは %z - タイムゾーンまたはその名前または短縮形。

%% - 文字リテラル '%'。

// 現在の時間を表示

```
var str = Date().toString("%Y / %m / %d %H:%M:%S");
```

- `exec(string jsCode)`  
与えられたコードを実行
- `eval(string jsCode)`  
与えられたコードを評価し結果を返す  

```
var code="1+2";  
var result = eval(code);
```
- `trace()`  
現在使用されている変数一覧をダンプする  

```
var variables = trace();
```
- `Object.dump()`  
変数の内容をダンプする  

```
var temp = ["1","2","3","4"]
```

## 関数一覧

```
temp.dump();
```

- `Object.clone()`  
Object をコピーする
- `Object.keys(object obj)`  
obj のインデックス一覧をオブジェクトに変換して返す
- `header(string str)`  
str を http ヘッダとして登録する
- `getLocalAddress()`  
ローカル IPV4 アドレスを返す。  
addr に現在の ipv4 アドレスを設定します。有線 LAN と無線 LAN など複数のローカル IP アドレスが存在する場合は外部と接続しやすいと思われるアドレスを取得します。  

```
var addr = getLocalAddress();
```
- `getLocalPort()`  
ローカルポートを返す  

```
var port = getLocalPort();
```
- `command(string path)`  
指定の OS コマンドを実行する。実行権限は Lutino.exe を起動したユーザーの権限となります。
- `htmlspecialchars(string url)`  

```
&:&amp;  
\:&quot;  
<:&lt;  
>:&gt;
```

  
url の特殊文字を上記ルールでエスケープする
- `encodeURI(string url)`  
「!\$()\*,-.:/?@[^\_`{}~\」以上の文字を%XX で数値表示にします。  
url をエンコードする

## 関数一覧

- `setCookie(string name,string value,string expire)`

`name=value` で指定されたクッキーを発行する。

有効期限は `expire` で指定された時間(秒)

0ならブラウザが開いている間のみ有効

これ以前に `print` する命令が出ていれば無効

```
setCookie('name','suzuki',60*60*48);
```

※48 時間分のクッキー

- `session_start()`

セッションを開始する。`setCookie` を使うので

これ以前に `print` する命令が出ていれば無効

- `shutdown(string password)`

全スクリプトを終了させる。`password` は `conf` ファイルで

指定する。指定してない場合には終了しない。

- `restful(string method,string url,string send_data)`

`method`:HEAD,GET,PUT,DELETE,POST

でリクエストを送る。

- `randomUUID()`

ランダムに生成された 36 文字の v4 UUID を格納した文字列。

```
var uuid = randomUUID();
```