

再考ですかーっ！？

JavaScript

SpiderMonkeyといっしょ

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

概要

- 📌 言語としてのJavaScriptをおさらい
 - 📌 基本的なSyntax
 - 📌 Object Oriented Language JavaScript
- 📌 JavaScriptの処理系を愉しむ
 - 📌 Mozillaの処理系”SpiderMonkey”概要
 - 📌 JavaScriptを俺色に染める

JavaScriptとは？

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

JavaScriptとは？



Netscape社がWebブラウザおよびサーバ用に開発したオブジェクト指向スクリプト言語

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

JavaScriptとは？

- 📌 Netscape社がWebブラウザおよびサーバ用に開発したオブジェクト指向スクリプト言語
- 📌 MicrosoftがマネしてJScriptとか
- 📌 結果Webブラウザによって実装ばらばら

JavaScriptとは？

- 📌 Netscape社がWebブラウザおよびサーバ用に開発したオブジェクト指向スクリプト言語
- 📌 MicrosoftがマネしてJScriptとか
 - 📌 結果Webブラウザによって実装ばらばら
- 📌 ECMAがECMAScript(ECMA-262 Edition3)として標準化し、ISO/IEC 16262に承認
 - 📌 各社ブラウザ、ソフトウェアが準拠
 - 📌 相変わらずDOMまわりは混沌としてるけど

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

C言語系Java風味の構文



- `var message = "Hello World!";`
- `if, else, else if, for, while, do while, switch`
- `continue, break, goto`
- `function name(arg1, arg2) { ... },`
`function (args) { ...}`
- `try, throw, catch, finally`
- `eval(...)`
- `+, -, *, %, ++, --...`
- `=, +=, -=, *=, %=...`
- `==, <, >, <=, >=, !=...`

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

Built-in Objects



Function
Array
String
Boolean
Number
Math
Date
RegExp

```
var msg = new String("Hello World!");  
var msg = "Hello" + " World!";
```

```
var list = new Array("Perl", "Java");  
list.length;  
list[1] += "Script";
```

```
var today = new Date();  
today.toString();
```

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

RegExp Object

 わりとモダンな正規表現

 String Objectにも関連メソッド多数

```
var reg = new RegExp("pattern", "gi");  
var reg = /pattern/gi;
```

¥b, ¥B, ¥s, ¥S

¥d, ¥D, ¥w, ¥W, .

[...], [^...]

*, +, ?, {n}, {n,}, {n, m}

^, \$

module.jp

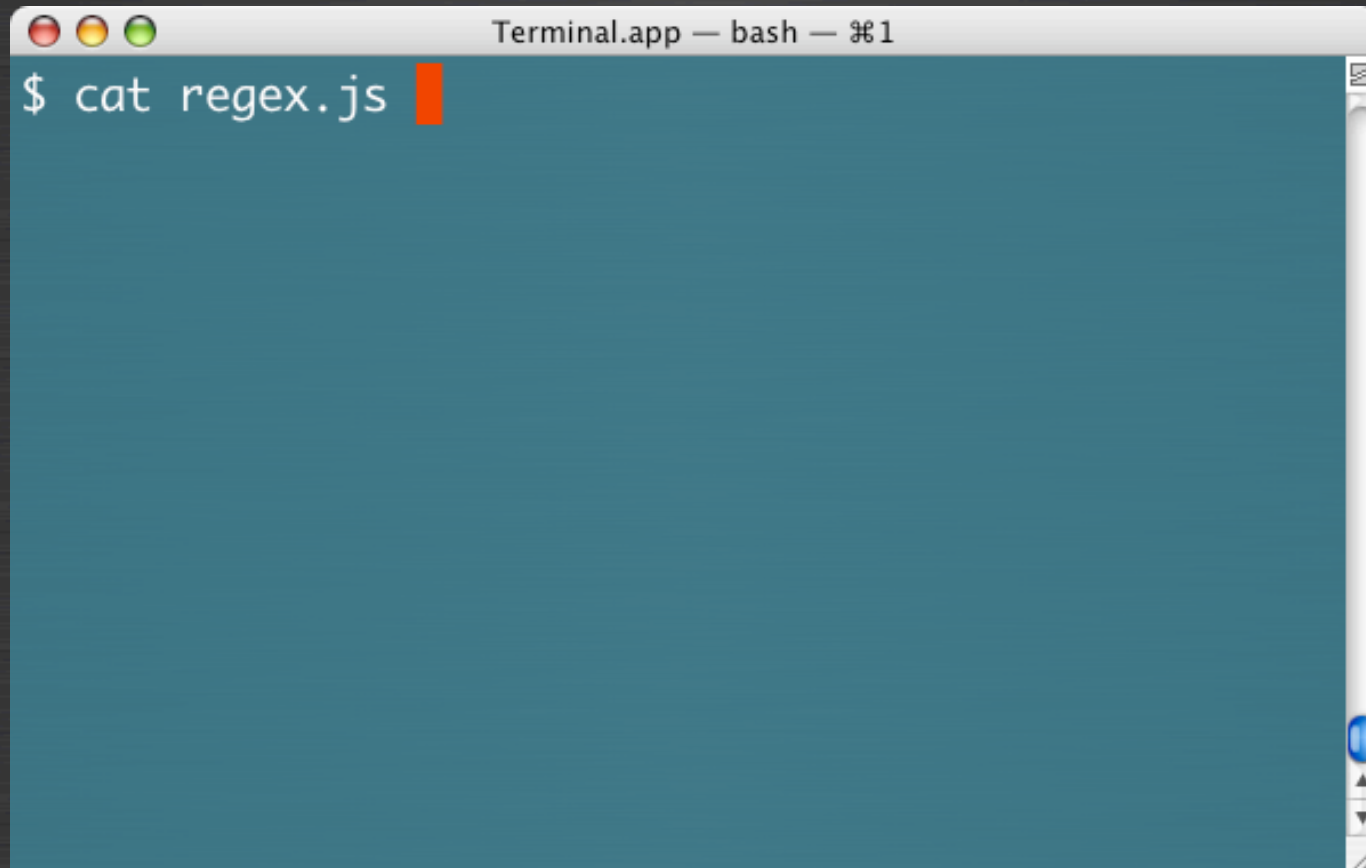
© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

RegExp Object

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

RegExp Object

A screenshot of a macOS Terminal window. The title bar at the top reads "Terminal.app — bash — ㉿1". The window has a teal background. The command prompt "\$ cat regex.js" is visible, followed by a red cursor. The right side of the window shows a vertical scrollbar and a blue button at the bottom.

```
Terminal.app — bash — ㉿1
$ cat regex.js
```

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

RegExp Object



```
Terminal.app — bash — ㉿1
$ cat regex.js
#!/usr/local/js/bin/js

var str = "177-0053:東京都練馬区関町南:小山浩之";
var list = str.split(/:/);
for (var i in list) {
    print("split: " + list[i]);
}
$
```

RegExp Object



```
Terminal.app — bash — ㉿1
$ cat regex.js
#!/usr/local/js/bin/js

var str = "177-0053:東京都練馬区関町南:小山浩之";
var list = str.split(/:/);
for (var i in list) {
    print("split: " + list[i]);
}
$ ./regex.js
```

RegExp Object

A terminal window titled "Terminal.app — bash — ㉿1" with a teal background. It shows the execution of a JavaScript file named "regex.js". The code defines a string "177-0053:東京都練馬区関町南:小山浩之" and splits it by colons. The output shows the string being split into three parts: "177-0053", "東京都練馬区関町南", and "小山浩之".

```
Terminal.app — bash — ㉿1
$ cat regex.js
#!/usr/local/js/bin/js

var str = "177-0053:東京都練馬区関町南:小山浩之";
var list = str.split(/:/);
for (var i in list) {
    print("split: " + list[i]);
}
$ ./regex.js
split: 177-0053
split: 東京都練馬区関町南
split: 小山浩之
$
```

module.jp

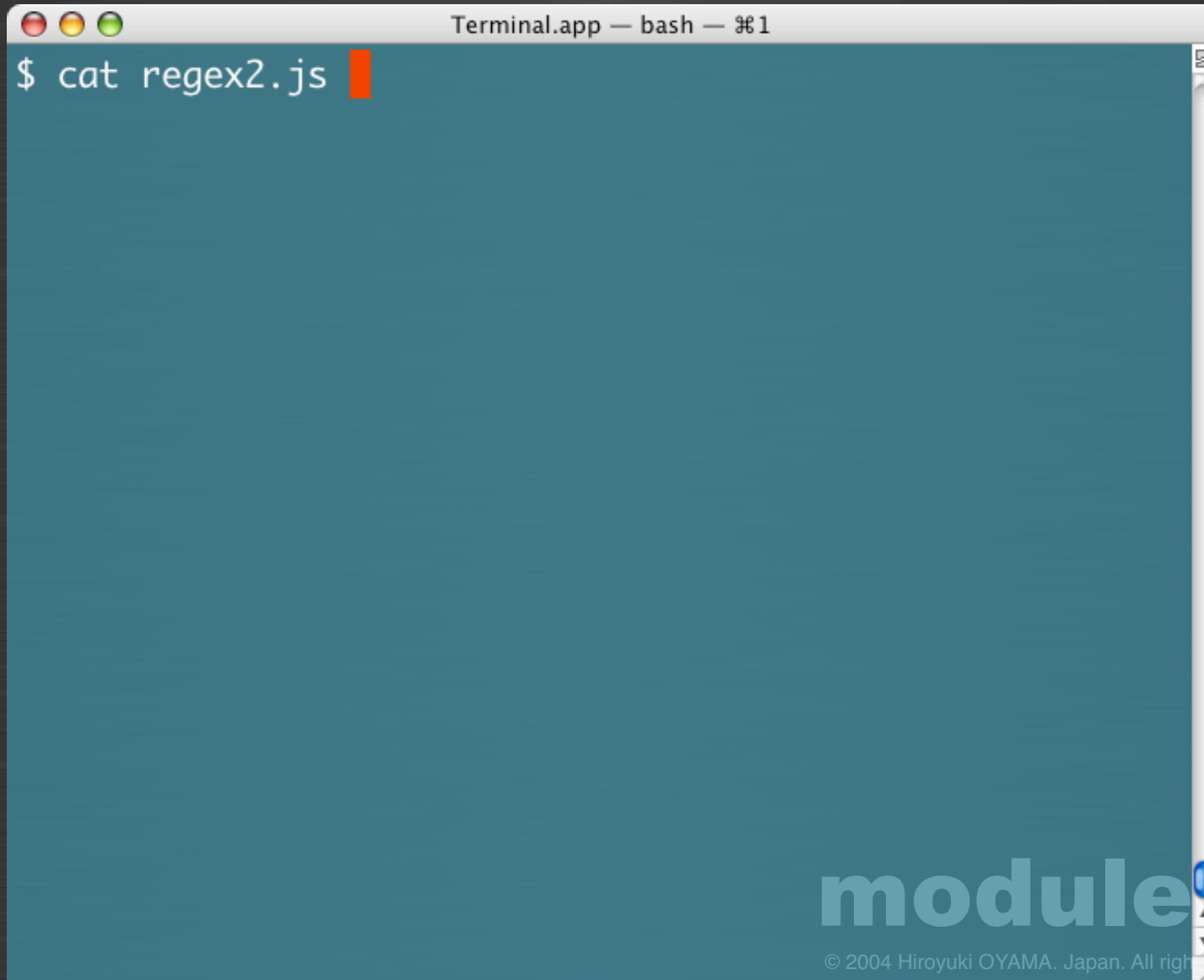
© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

RegExp Object

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

RegExp Object



modulejp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

RegExp Object

```
Terminal.app — bash — ㉿1
$ cat regex2.js
#!/usr/local/js/bin/js

var reg = new RegExp("([\x21-\x7E]+|"
                    + "([\xA4[\xA1-\xF3]])+|"
                    + "([\xA5[\xA1-\xF6]])+|"
                    + "([\xB0-\xF4][\x00-\xFF])+)");
var str = "漢字カナASCIIまじりの文字列";
var words = str.match(reg);
for (var i in words) {
    print(words[i]);
}
$
```

RegExp Object

```
Terminal.app — bash — ㊟1
$ cat regex2.js
#!/usr/local/js/bin/js

var reg = new RegExp("([\x21-\x7E]+| "
                    + "([\xA4[\xA1-\xF3])| "
                    + "([\xA5[\xA1-\xF6])| "
                    + "([\xB0-\xF4][\x00-\xFF])+) ", "g");
var str = "漢字カナASCIIまじりの文字列";
var words = str.match(reg);
for (var i in words) {
    print(words[i]);
}
$ ./regex2.js
```

RegExp Object

```
Terminal.app — bash — ㊦1
$ cat regex2.js
#!/usr/local/js/bin/js

var reg = new RegExp("([\x21-\x7E]+|"
                    + "([\xA4[\xA1-\xF3]])+|"
                    + "([\xA5[\xA1-\xF6]])+|"
                    + "([\xB0-\xF4][\x00-\xFF])+)");
var str = "漢字カナASCIIまじりの文字列";
var words = str.match(reg);
for (var i in words) {
    print(words[i]);
}
$ ./regex2.js
漢字
カナ
ASCII
まじりの
文字列
$
```

Object Oriented JavaScript

- 📌 プロトタイプベースのオブジェクト指向言語
- 📌 継承ではなくて委譲

```
SubClass.prototype = new SuperClass();  
function SubClass () {  
    this.myMethod = function () {  
        print("Hello World!");  
    }  
}
```

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

SpiderMonkey

- SpiderMonkey (JavaScript-C) Engine
- MozillaのJavaScript実装
 - 単体のパッケージとして利用可能
 - Netscape Enterprise Server
 - K-3D (3Dモデリング・アニメーション)
 - スタンドアローンのインタプリタが付属
 - Fileオブジェクト (独自拡張Class)
 - JavaとPerlのBinding付属

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

SpiderMonkey配布元



<http://www.mozilla.org/js/spidermonkey/>

SpiderMonkey (JavaScript-C) Engine

What is SpiderMonkey?

SpiderMonkey is the code-name for the Mozilla's C implementation of JavaScript.

Where do I get it?

The core SpiderMonkey engine can be found in [mozilla/js/src](#). The stand alone interpreter can be built using [Makefile.ref](#). Read [mozilla/js/src/README.html](#) for the nitty gritty. Some projects embedding the JavaScript engine (in addition to Mozilla itself) are listed in the [projects](#) page.

You can get the engine via [CVS](#) and [build it yourself](#), or look for recent tarballs at (please check the [mirrors](#) first), <http://ftp.mozilla.org/pub/mozilla.org/js/>. Release notes are available at <http://www.mozilla.org/js/spidermonkey/release-notes/>.

Where do I find out more?

Site	Description
JS Embedder's Guide	A guide to embedding the JavaScript C engine in applications.
JS Embedder's Reference in the following formats: One entry at a time	A function by function reference to the public JavaScript API

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

ビルド手順

- 📌 Mozilla用のビルドシステムなので使いにくい
 - 📌 プラットフォーム決め撃ち条件コンパイル
 - 📌 自分でLibtoolを作らない
- 📌 勝手にGNU Autotools化してしまえ！
 - 📌 configure.in
 - 📌 Makefile.am
 - 📌 aclocal & libtoolize & automake & autoheader & autoconf

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

ビルド手順

http://module.jp/blog/autotoolize_spidermonkey.html

configureでプラットフォームを判定して、#defineしてやる

AC_DEFINE(XP_UNIX)

AC_DEFINE(SYS4)

AC_DEFINE(SYSV)

AC_DEFINE(_BSD_SOURCE)

AC_DEFINE(POSIX_SOURCE)

AC_DEFINE(DARWIN)



The screenshot shows the module.jp website with the following content:

- Header: module.jp Apache & Perl Module Information. Navigation: Home, Book, Cookbook, Blog.
- Blog Post: "SpiderMonkeyをGNU Autotools対応する" by Hiroyuki OYAMA, dated Tue Oct 12 02:11:52 2004.
- Main Text: "MozillaブラウザのJavaScriptエンジンであるSpiderMonkeyを他のプロジェクトでもサクッと使えるように、GNU Autoconf, Automake, Libtoolでビルドできるように修正するお話。"
- Left Sidebar: "PDF印刷" (PDF printing) and "PDFは印刷に最適 誰でも出来るPDF作成講座" (PDF is optimal for printing, a course for anyone to learn PDF creation).
- Right Sidebar: "About Us" with links to "このサイトについて" (About this site), "小山浩之について" (About Hiroyuki Oyamada), "著作・雑誌記事など" (Works, magazine articles, etc.), and "RSS". Below it is an advertisement for "Apacheモジュールプログラミングガイド" (Apache Module Programming Guide) by Hiroyuki Oyamada, published by No Starch Press.

module.jp

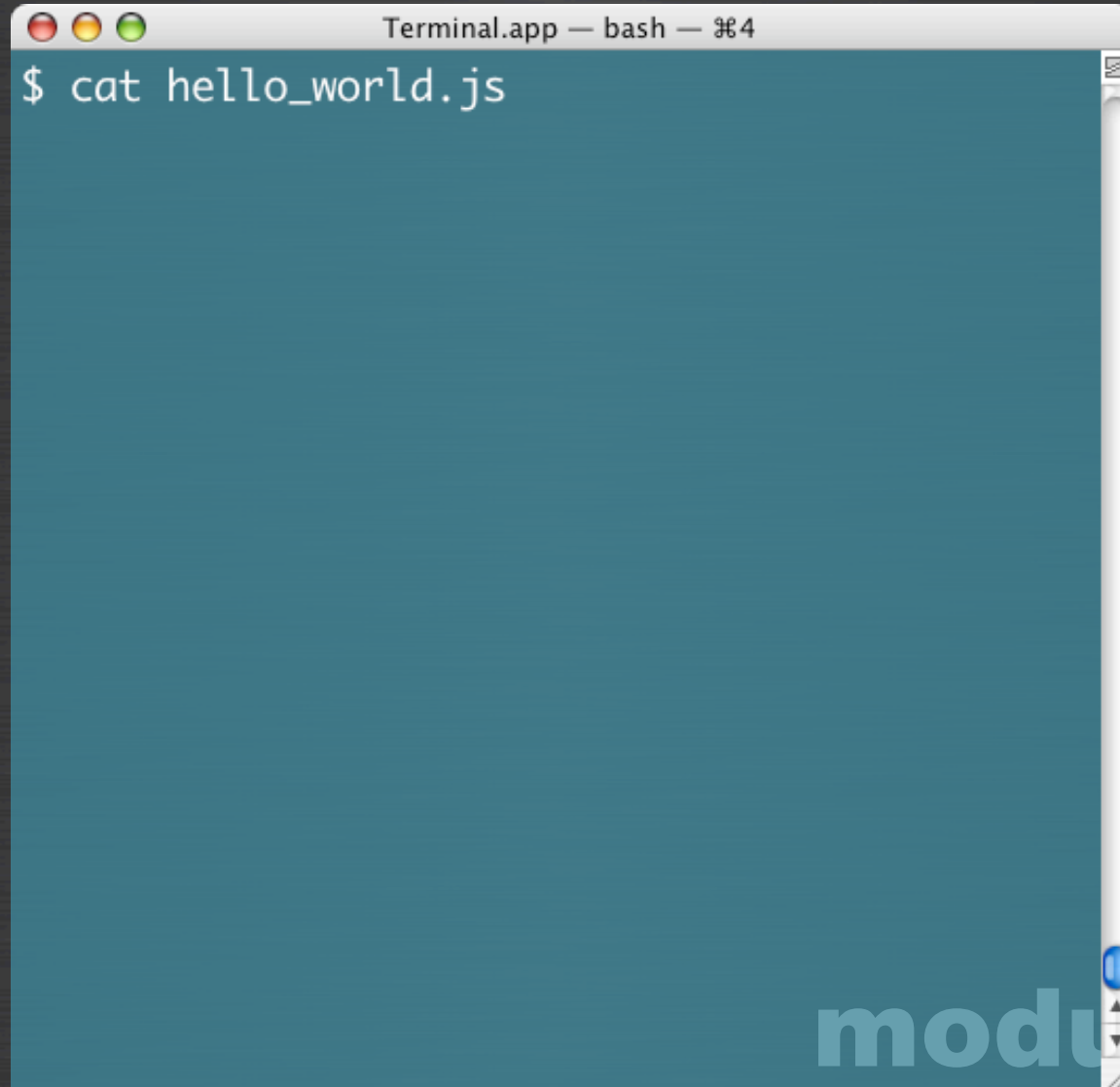
© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

スタンドアローンのインタプリタ

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

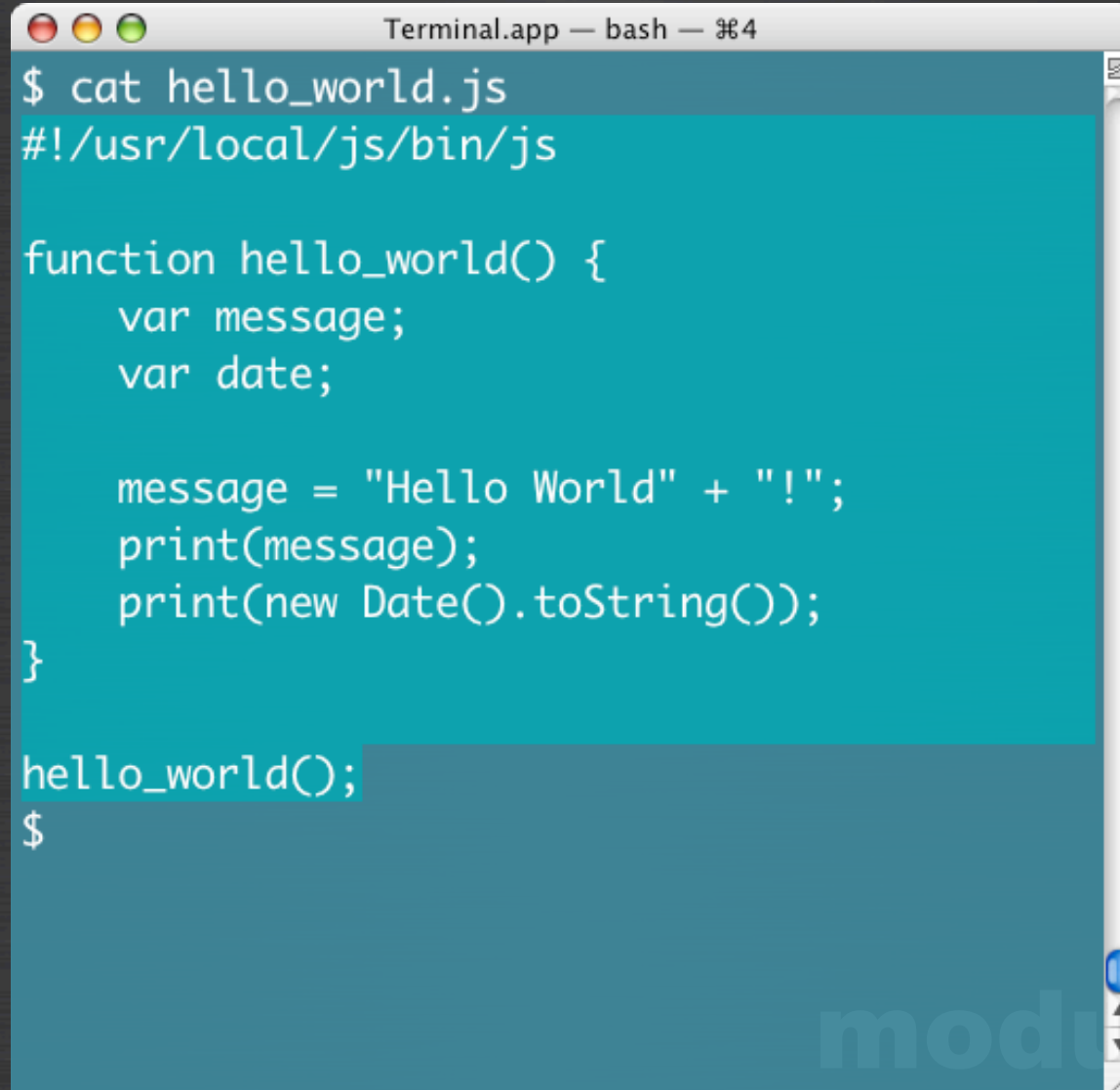
スタンドアローンのインタプリタ



module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

スタンドアローンのインタプリタ

A screenshot of a macOS Terminal window titled "Terminal.app — bash — ㊿4". The window has a teal background and displays JavaScript code. The code defines a function named "hello_world()" which declares two variables, "message" and "date". It then assigns a string "Hello World" followed by an exclamation mark to "message", and prints both "message" and the current date using "print(message)" and "print(new Date().toString())". Finally, the function is called with "hello_world()". The prompt "\$" is visible at the bottom.

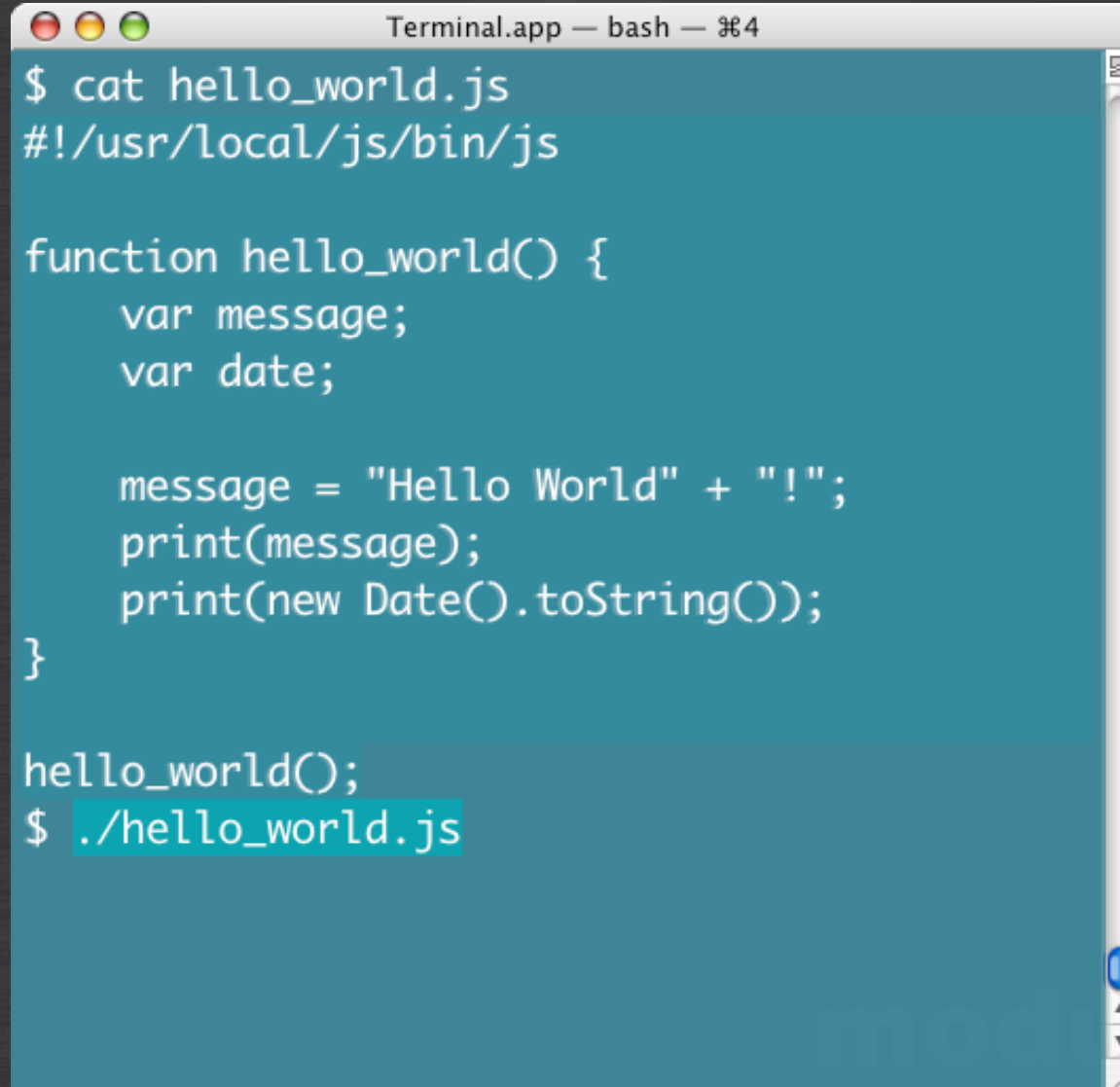
```
Terminal.app — bash — ㊿4
$ cat hello_world.js
#!/usr/local/js/bin/js

function hello_world() {
    var message;
    var date;

    message = "Hello World" + "!";
    print(message);
    print(new Date().toString());
}

hello_world();
$
```

スタンドアローンのインタプリタ

A screenshot of a macOS Terminal window titled "Terminal.app — bash — ㊿4". The window has a teal background. The code shown is a JavaScript file named "hello_world.js". It defines a function "hello_world()" that sets a message, prints it, and prints the current date. The function is then called. The command to run the file is highlighted in blue.

```
Terminal.app — bash — ㊿4
$ cat hello_world.js
#!/usr/local/js/bin/js

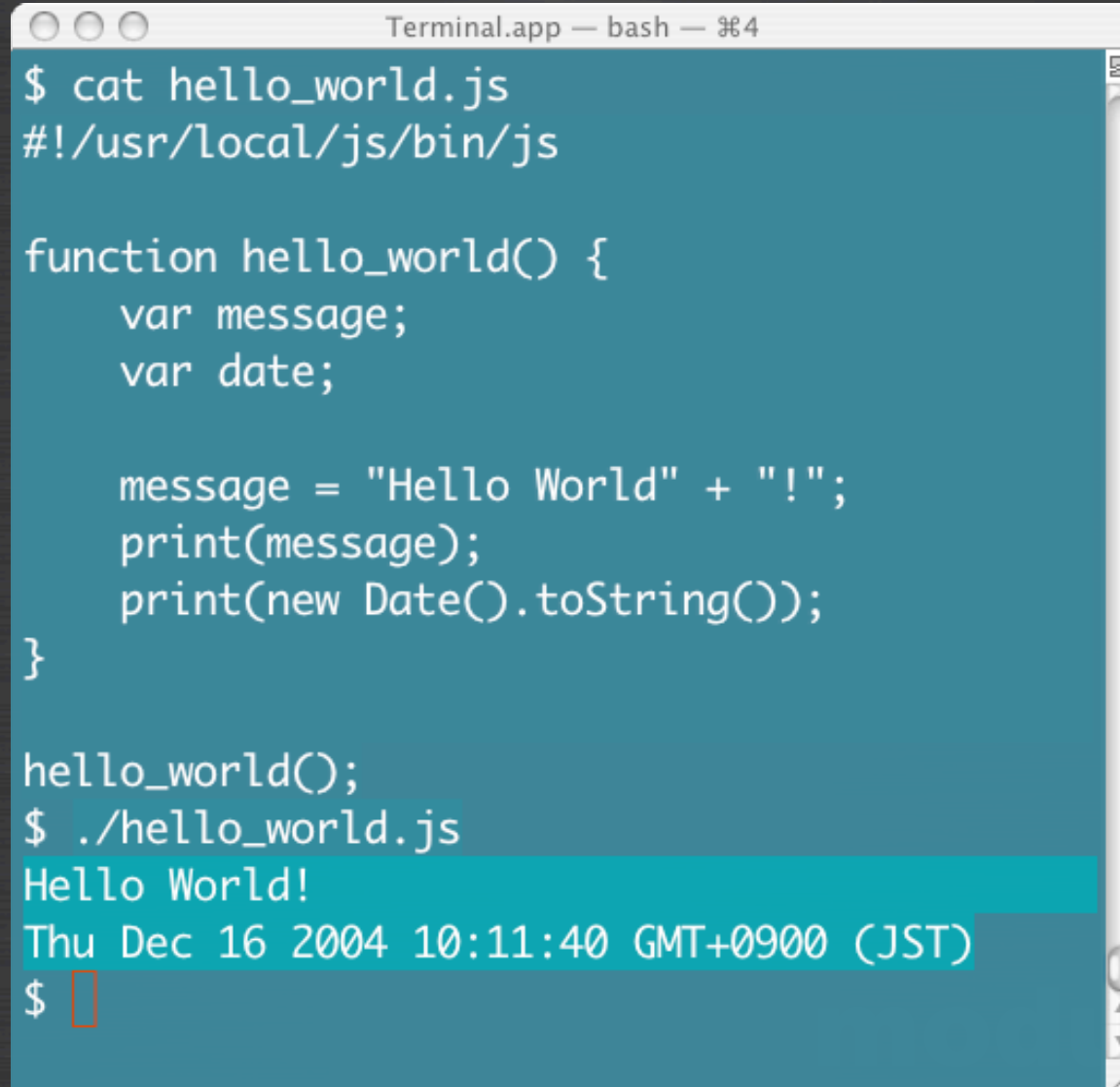
function hello_world() {
    var message;
    var date;

    message = "Hello World" + "!";
    print(message);
    print(new Date().toString());
}

hello_world();
$ ./hello_world.js
```

le.jp

スタンドアローンのインタプリタ



```
Terminal.app — bash — ㉿4
$ cat hello_world.js
#!/usr/local/js/bin/js

function hello_world() {
    var message;
    var date;

    message = "Hello World" + "!";
    print(message);
    print(new Date().toString());
}

hello_world();
$ ./hello_world.js
Hello World!
Thu Dec 16 2004 10:11:40 GMT+0900 (JST)
$ 
```

le.jp

とりあえずCから使うには？

- 📌 **jsapi.h**をincludeせよ！
- 📌 ランタイムとコンテキストと標準クラスの初期化
- 📌 ***JSRuntime*** *rt = **JS_NewRuntime**(*RUNTIME_SIZE*);
- 📌 ***JSContext*** *ctx = **JS_NewContext**(rt, *STACK_SIZE*);
- 📌 ***JSObject*** *global = **JS_NewObject**(ctx, &*GLOBAL_CLASS*, *NULL*, *NULL*);
- 📌 **JS_InitStandardClasses**(ctx, *global*);
- 📌 **libjs**をリンクせよ！

スクリプトを実行するには？

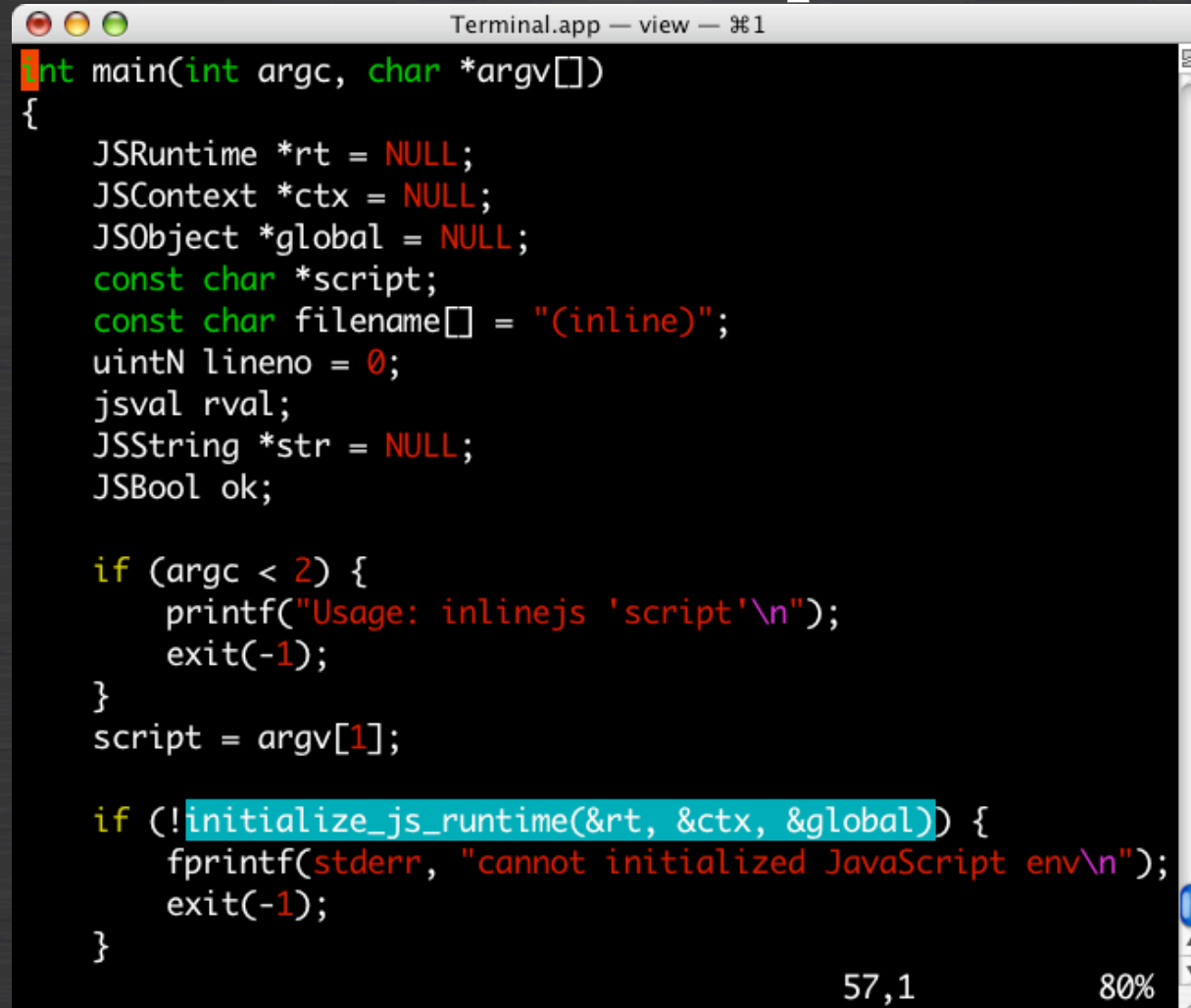
- 📌 JSContextとJSObject(のglobal object)を用意して、evalする。戻り値は***jsval*** rvalで取得する。
- 📌 **JS_EvaluateScript**(ctx, global, SCRIPT_STR, strlen(SCRIPT_STR), filename, linenum, &rval);
- 📌 **JSString** *str = **JS_ValueToString**(ctx, rval);
- 📌 printf("result: '%s'", **JS_GetStringBytes**(str));
- 📌 **JS_DestroyContext**(ctx); /* あとしまつ1 */
- 📌 **JS_DestroyRuntime**(rt); /* あとしまつ2 */

CからJavaScriptを呼び出す

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

CからJavaScriptを呼び出す

A screenshot of a macOS Terminal window titled "Terminal.app — view — ㊟1". The window displays a C program that initializes a JavaScript runtime and calls a function named "inlinejs". The code is color-coded: keywords in blue, identifiers and literals in green, and string literals in red. The program includes headers for JavaScript runtime and standard C functions. It checks for command-line arguments and prints usage if there are fewer than 2 arguments. It then initializes the JavaScript runtime and calls "inlinejs" with the script content. The code is as follows:

```
int main(int argc, char *argv[])
{
    JSRuntime *rt = NULL;
    JSContext *ctx = NULL;
    JSObject *global = NULL;
    const char *script;
    const char filename[] = "(inline)";
    uintN lineno = 0;
    jsval rval;
    JSString *str = NULL;
    JSBool ok;

    if (argc < 2) {
        printf("Usage: inlinejs 'script'\n");
        exit(-1);
    }
    script = argv[1];

    if (!initialize_js_runtime(&rt, &ctx, &global)) {
        fprintf(stderr, "cannot initialized JavaScript env\n");
        exit(-1);
    }
}
```

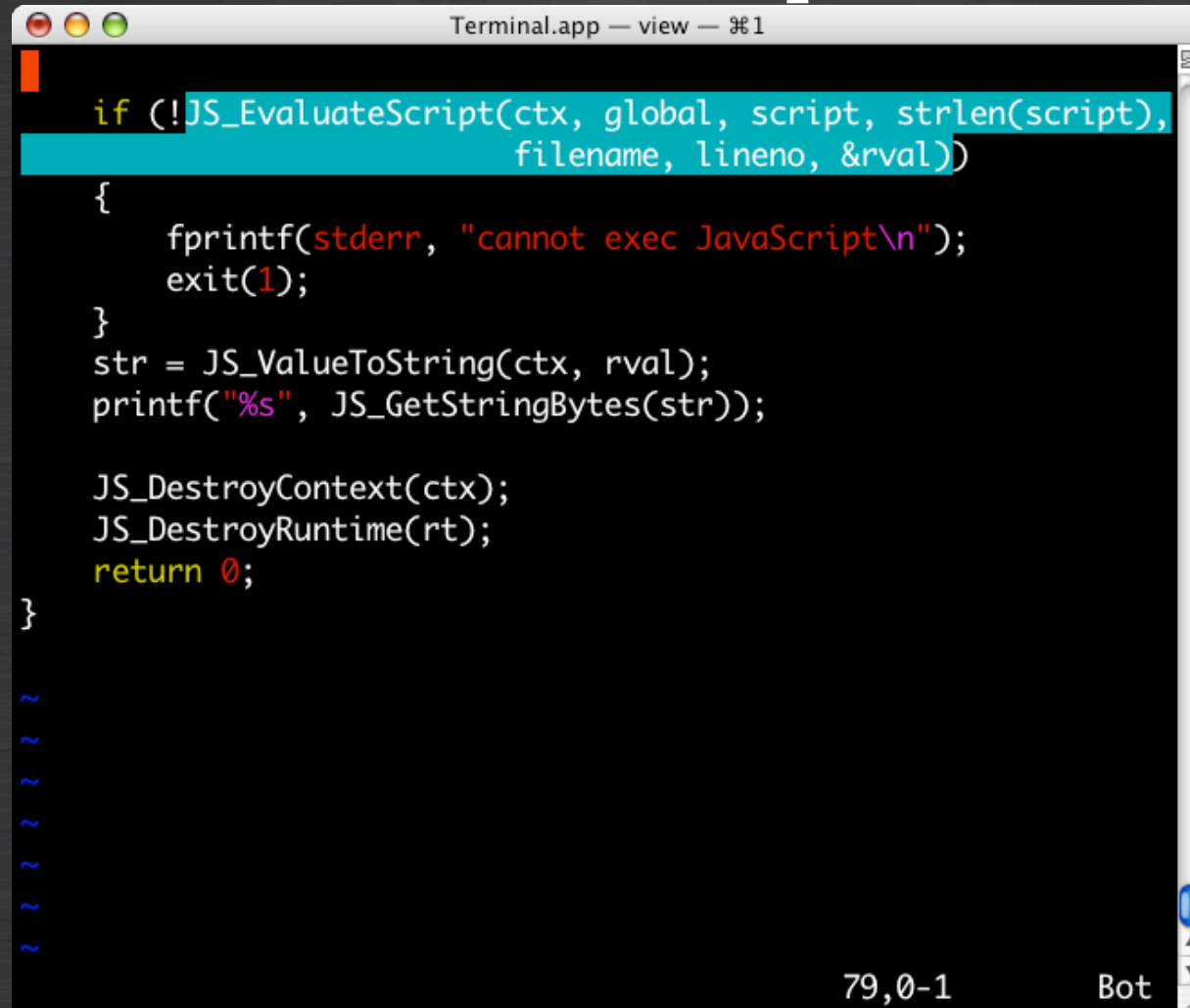
57,1

80%

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

CからJavaScriptを呼び出す

A screenshot of a macOS Terminal window titled "Terminal.app — view — ㉿1". The window displays a C code snippet with syntax highlighting. The code is enclosed in a function and checks if JavaScript evaluation fails. If it fails, it prints an error message to stderr and exits with status 1. If successful, it converts the JavaScript result to a C string and prints it. Finally, it destroys the JavaScript context and runtime before returning 0. The code is as follows:

```
if (!JS_EvaluateScript(ctx, global, script, strlen(script),  
                        filename, lineno, &rval))  
{  
    fprintf(stderr, "cannot exec JavaScript\n");  
    exit(1);  
}  
str = JS_ValueToString(ctx, rval);  
printf("%s", JS_GetStringBytes(str));  
  
JS_DestroyContext(ctx);  
JS_DestroyRuntime(rt);  
return 0;  
}
```

At the bottom of the terminal window, the text "79,0-1" and "Bot" are visible.

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

独自関数の追加

- 📌 インタプリタにCの関数を登録してJavaScriptから実行できる。
- 📌 JSBool **myfunc**(JSContext *ctx, JSObject *obj, uintN argc, jsval *argv, jsval *rval);
- 📌 JSFunctionSpec構造体(の配列)を定義
 - 📌 関数名(JavaScript側での関数名)
 - 📌 呼び出す関数のポインタ(myfunc)
 - 📌 引数の数
- 📌 JS_DefineFunctions()で、JSFunctionSpec構造体を登録

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

拡張例 - `save_to_file()`

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

拡張例 - save to file()

```
Terminal.app — vim — ㉿1

#include <stdio.h>
#include <jsapi.h>

static JSBool my_save_to_file(JSContext *ctx, JSObject *obj,
                             uintN argc, jsval *argv, jsval *rval)
{
    const char *fname, *text;
    FILE *fp;

    fname = JS_GetStringBytes(JS_ValueToString(ctx, argv[0]));
    text = JS_GetStringBytes(JS_ValueToString(ctx, argv[1]));

    fp = fopen(fname, "w");
    fprintf(fp, text);

    return JS_TRUE;
}

static JSFunctionSpec my_functions[] = {
    { "save_to_file", my_save_to_file, 2, 0, 0 },
    { 0, 0, 0, 0, 0 },
};

"save.c" 115L, 2700C written
```

1,1

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

拡張例 - save to file()

```
Terminal.app — vim — ㉿1
#include <stdio.h>
#include <jsapi.h>

static JSBool my_save_to_file(JSContext *ctx, JSObject *obj,
                             uintN argc, jsval *argv, jsval *rval)
{
    const char *fname, *text;
    FILE *fp;

    fname = JS_GetStringBytes(JS_ValueToString(ctx, argv[0]));
    text = JS_GetStringBytes(JS_ValueToString(ctx, argv[1]));

    fp = fopen(fname, "w");
    fprintf(fp, text);

    return JS_TRUE;
}

static JSFunctionSpec my_functions[] = {
    { "save_to_file", my_save_to_file, 2, 0, 0 },
    { 0, 0, 0, 0, 0 },
};

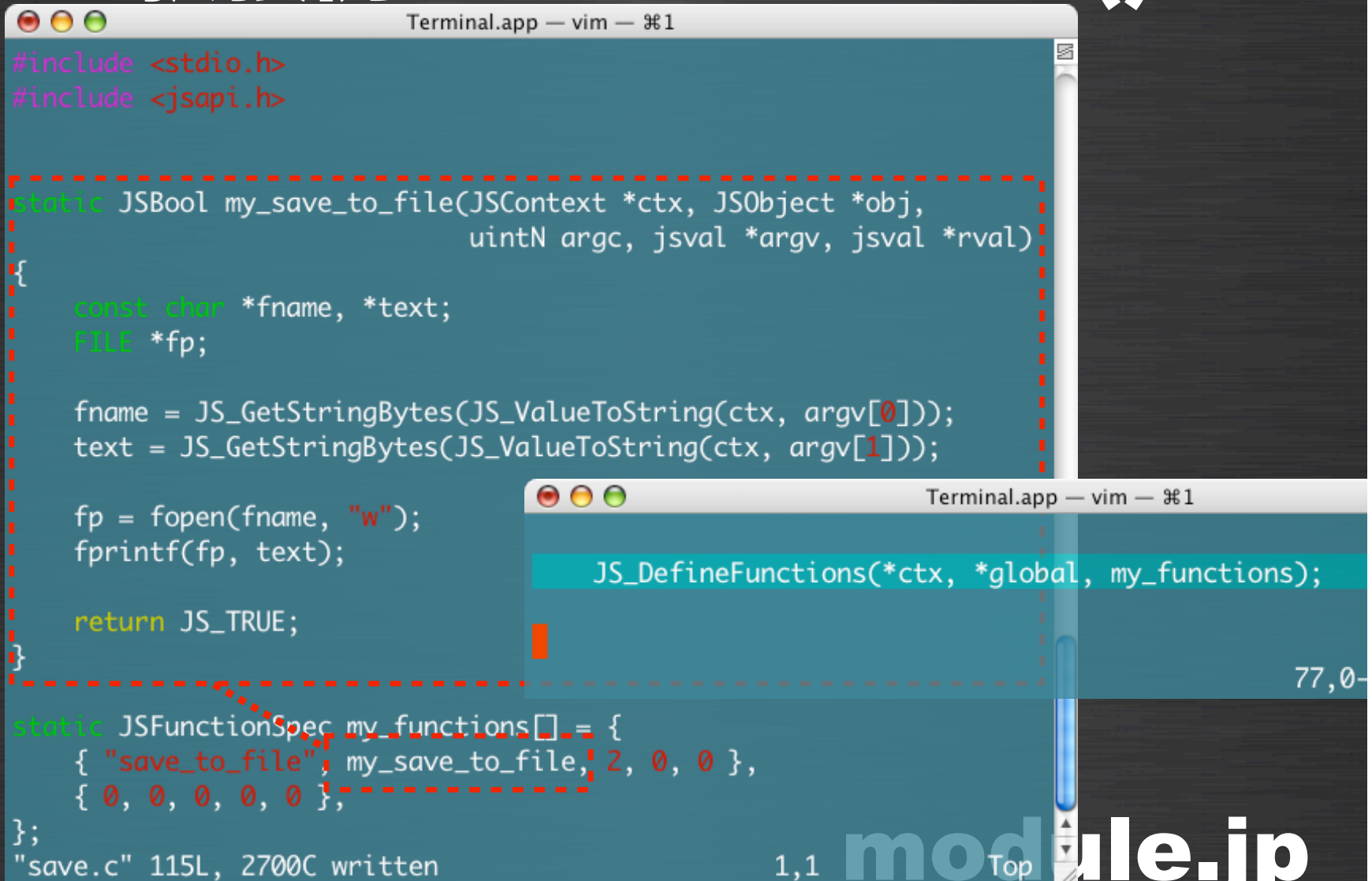
"save.c" 115L, 2700C written
```

1,1

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

拡張例 - save to file()



```
Terminal.app — vim — ㉿1
#include <stdio.h>
#include <jsapi.h>

static JSBool my_save_to_file(JSContext *ctx, JSObject *obj,
                             uintN argc, jsval *argv, jsval *rval)
{
    const char *fname, *text;
    FILE *fp;

    fname = JS_GetStringBytes(JS_ValueToString(ctx, argv[0]));
    text = JS_GetStringBytes(JS_ValueToString(ctx, argv[1]));

    fp = fopen(fname, "w");
    fprintf(fp, text);

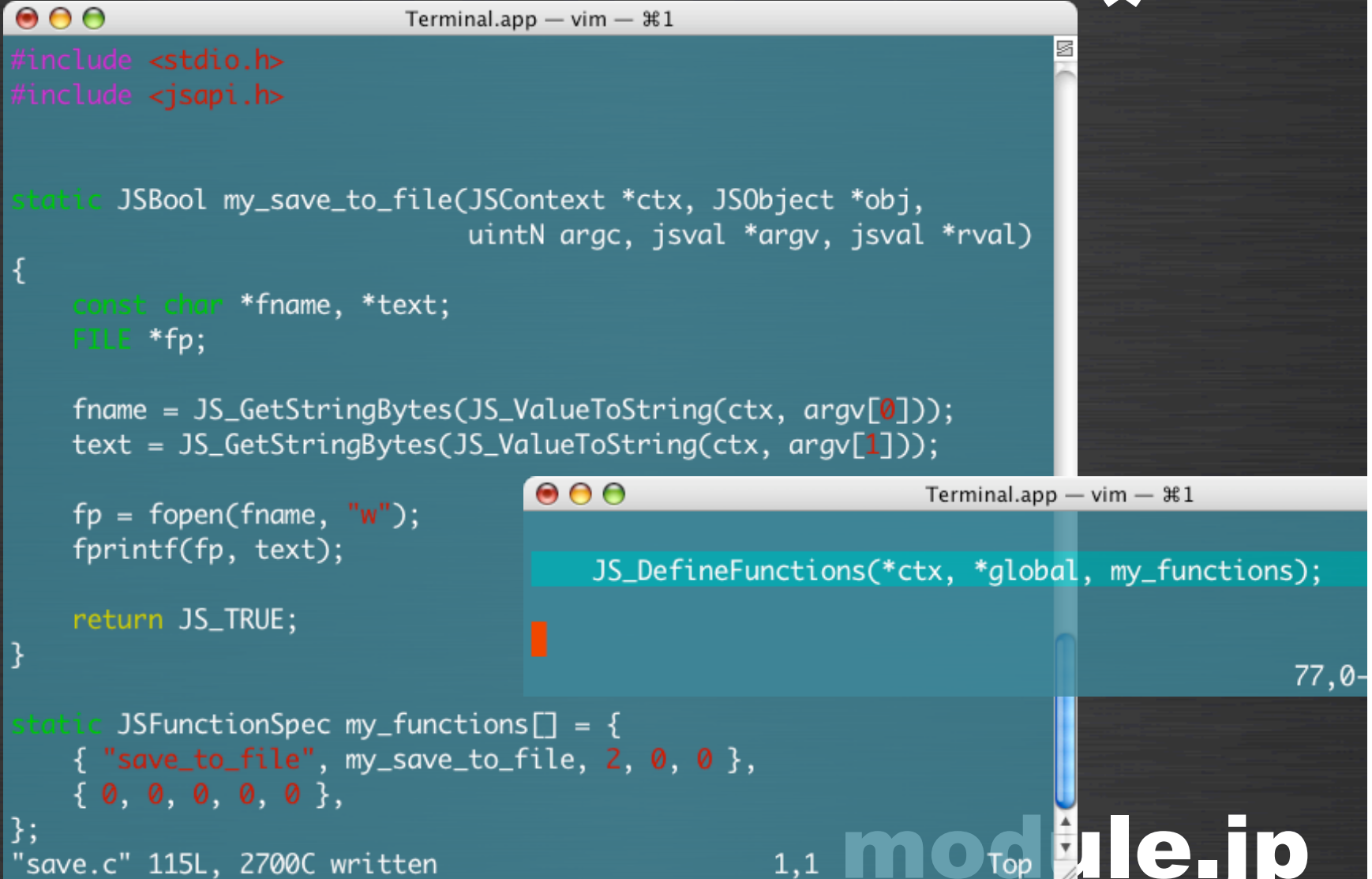
    return JS_TRUE;
}

static JSFunctionSpec my_functions[] = {
    { "save_to_file", my_save_to_file, 2, 0, 0 },
    { 0, 0, 0, 0, 0 },
};

"save.c" 115L, 2700C written
```

```
Terminal.app — vim — ㉿1
JS_DefineFunctions(*ctx, *global, my_functions);
77,0-
```

拡張例 - save to file()



```
Terminal.app — vim — ㉿1
#include <stdio.h>
#include <jsapi.h>

static JSBool my_save_to_file(JSContext *ctx, JSObject *obj,
                             uintN argc, jsval *argv, jsval *rval)
{
    const char *fname, *text;
    FILE *fp;

    fname = JS_GetStringBytes(JS_ValueToString(ctx, argv[0]));
    text = JS_GetStringBytes(JS_ValueToString(ctx, argv[1]));

    fp = fopen(fname, "w");
    fprintf(fp, text);

    return JS_TRUE;
}

static JSFunctionSpec my_functions[] = {
    { "save_to_file", my_save_to_file, 2, 0, 0 },
    { 0, 0, 0, 0, 0 },
};

"save.c" 115L, 2700C written

JS_DefineFunctions(*ctx, *global, my_functions);
77,0-
```

1,1

module.jp

拡張例 - save to file()

```
#include <stdio.h>
#include <jsapi.h>

static JSBool my_save_to_file(JSContext *ctx, JSObject *obj,
                             uintN argc, jsval *argv, jsval *rval)
{
    const char *fname, *text;
    FILE *fp;

    fname = JS_GetStringBytes(JS_ValueToString(ctx, argv[0]));
    text = JS_GetStringBytes(JS_ValueToString(ctx, argv[1]));

    fp = fopen(fname, "w");
    fprintf(fp, text);

    return JS_TRUE;
}

static JSFunctionSpec my_functions[] = {
    { "save_to_file", my_save_to_file, 2, 0, 0 },
    { 0, 0, 0, 0, 0 },
};

"save.c" 115L, 2700C written
```

Terminal.app — vim — ㉿1

Terminal.app — vim — ㉿1

JS_DefineFunctions(*ctx, *global, my_functions);

77,0-

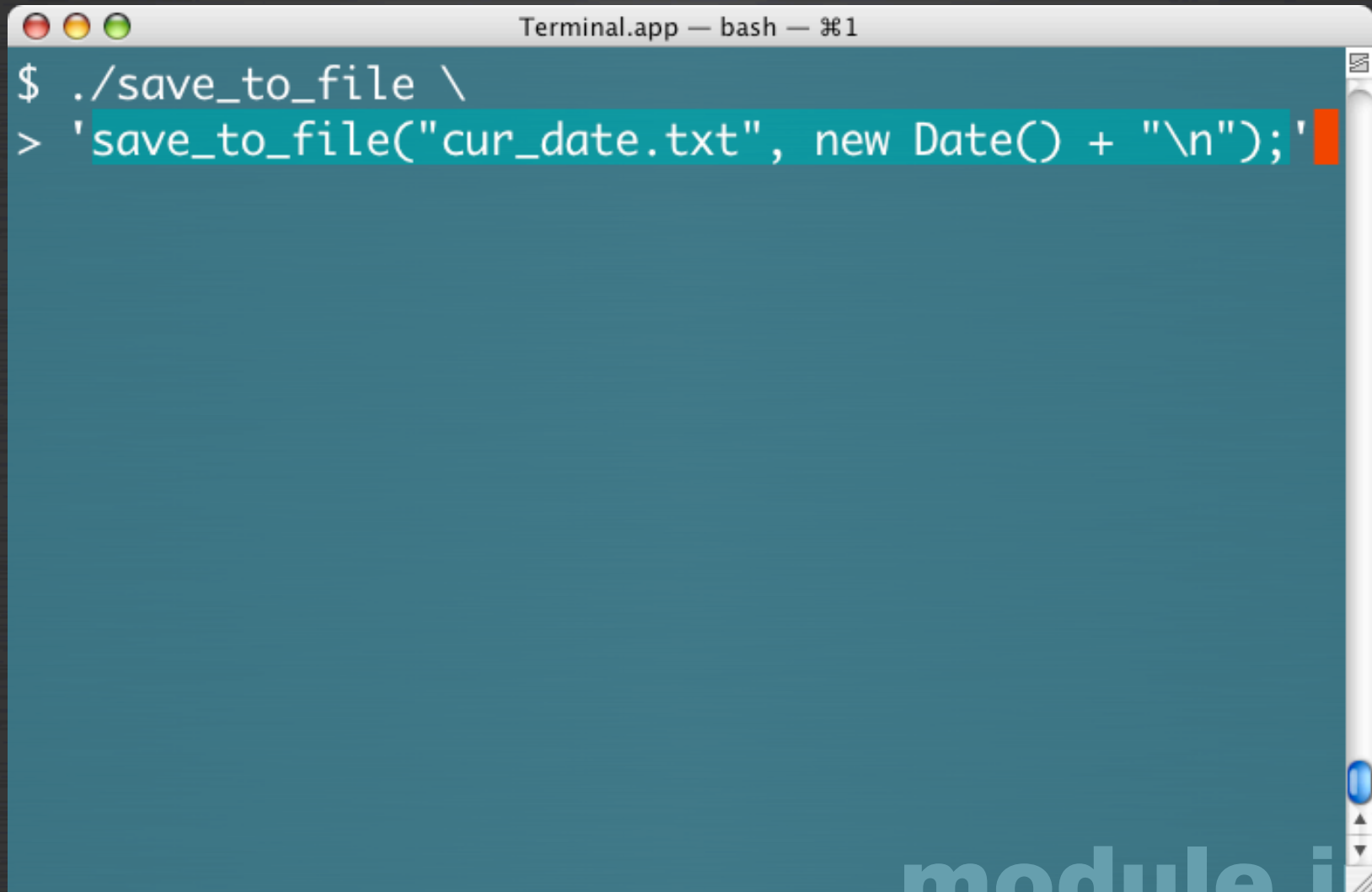
1,1 module.jp

拡張例 - save_to_file()

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

拡張例 - save_to_file()

A screenshot of a macOS Terminal window. The title bar reads "Terminal.app — bash — ㏻1". The terminal has a teal background. The prompt "\$./save_to_file \" is on the first line. The second line shows a command being entered: "> 'save_to_file(\"cur_date.txt\", new Date() + \"\\n\");'". The text after the prompt is highlighted in light blue. A red cursor is at the end of the command. On the right side of the terminal, there is a vertical scrollbar and a small icon at the top.

```
$ ./save_to_file \  
> 'save_to_file("cur_date.txt", new Date() + "\\n");'
```

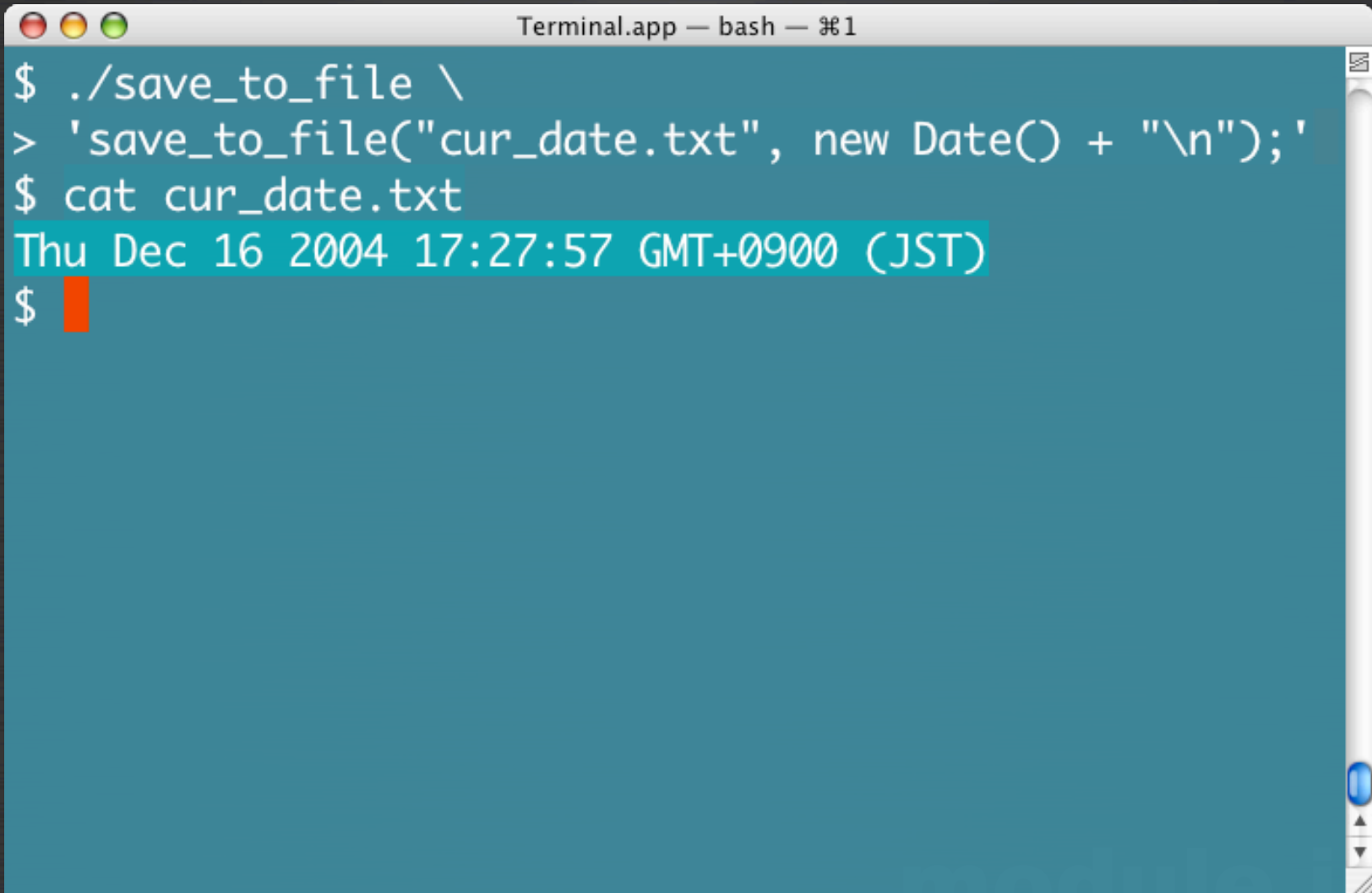

拡張例 - save_to_file()

A screenshot of a macOS Terminal window titled "Terminal.app — bash — ㏿1". The window has a teal background and a white title bar with standard macOS window controls. The terminal shows the following commands and their output:

```
$ ./save_to_file \  
> 'save_to_file("cur_date.txt", new Date() + "\n");'  
$ cat cur_date.txt
```

The second line of the command is highlighted in light blue. The terminal window is positioned over a dark background.

拡張例 - save_to_file()



```
Terminal.app — bash — ㏿1
$ ./save_to_file \
> 'save_to_file("cur_date.txt", new Date() + "\n");'
$ cat cur_date.txt
Thu Dec 16 2004 17:27:57 GMT+0900 (JST)
$
```

利用例

-  Apache moduleでPHPみたいなやつ
-  HTML側にJavaScriptで記述したvalidationロジックを、サーバ側で実行・検証する。
-  アプリケーションのカスタマイズ用埋込言語

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

再考してみると？

- 📌 ナニゲに面白いぞJavaScript
 - 📌 JavaScript悪くない。悪いはブラウザ
 - 📌 イジリ甲斐があるぞJavaScript
 - 📌 OSSの処理系が手に入るぞJavaScript
 - 📌 拡張し甲斐があるぞJavaScript
 - 📌 組み込みたくなっちゃうぞJavaScript
 - 📌 書ける人 or 書けそうな人って多いよね
- JavaScript

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

再考ですかーっ！？

JavaScript

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

最高ですかーっ！？

JavaScript

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.

小山浩之 Hiroyuki OYAMA

〒177-0053

東京都練馬区関町南4-17-1 106

Mail: oyama@module.jp

Web: <http://module.jp/>

module.jp

© 2004 Hiroyuki OYAMA. Japan. All rights reserved.