A vibrant sunset over the Portland, Oregon skyline, featuring the Moda Center, the Hawthorne Bridge, and the Willamette River.

# node.js INTERACTIVE



# Carpentry with Duktape

Tyler Brock, Hustle Inc.

# JS Culture

A wide-angle photograph of a city skyline during sunset or sunrise. The sky is filled with warm, orange, and yellow hues. The buildings in the foreground and middle ground are brightly lit, their windows reflecting the colorful light. The overall atmosphere is dynamic and modern.

MORE  
JAVASCRIPT!



# What is Duktape?

- Embeddable ECMAScript 5 Engine
  - Extend functionality of C/C++ code through flexible scripting
  - Leverage speedy C functions from within JavaScript program

# Features

- ES5 and 5.1 support
- Tiny amount of ES6 (proxy object subset, setPrototypeOf, ...)
- Some browser enhancements (like print + alert)
- Built in debugger
- CommonJS, regex, and unicode support (cross platform)

# Project Goals

- Compliance
- Portability
- Easy C Interface
- Small footprint
- Reasonable Performance (ASCII String Performance)

# What do I need?

- duktape.h (provides API)
- duktape.c (provides implementation)
- duk\_config.h (configuration -- included by duktape.h)

# How do I use it?

```
#include "duktape.h"

int main() {
    duk_context *ctx = duk_create_heap_default();
    duk_eval_string(ctx, "print('Hello world!');");
    duk_destroy_heap(ctx);
}
```

# Leverage JS flexibility from C/C++

```
function cool(str) {  
    print('working magic...');  
    var coolerStr = str + " is cool";  
    return coolerStr;  
}
```

## duk\_eval\_file()

### Prototype

```
void duk_eval_file(duk_context *ctx, const char *path);
```

### Stack



### Summary

Like [duk\\_eval\(\)](#), but the eval input is given as a filename. The filename associated with the temporary function created for the eval code is path as is.

**NOTE:** The path argument is given directly to `fopen()`. The path encoding cannot be specified and there is no support for a wide character string.

### Example

```
duk_eval_file(ctx, "test.js");
printf("result is: %s\n", duk_safe_to_string(ctx, -1));
duk_pop(ctx);
```

### See also

- [duk\\_eval\\_file\\_noresult](#)

# duk\_eval\_file()

compile

nonportable

1.0.0

## Prototype

```
void duk_eval_file(duk_context *ctx, const char *path);
```

C

## Stack



## Summary

Like [duk\\_eval\(\)](#), but the eval input is given as a filename. The filename associated with the temporary function created for the eval code is path as is.

# duk\_push\_string()

stack

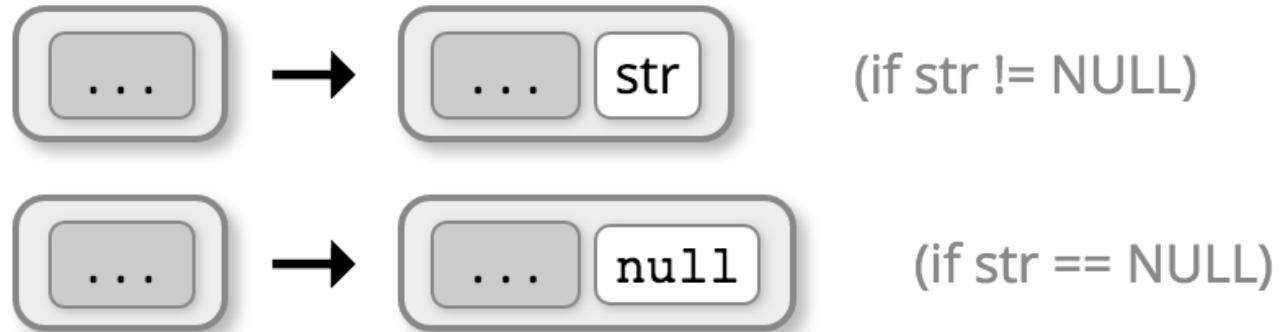
string

1.0.0

Prototype

```
const char *duk_push_string(duk_context *ctx, const char *str);
```

Stack



Summary

Push a C string into the stack. String length is automatically detected with a `strlen()` equivalent (i.e. looking for the first NUL character). A pointer to the interned string data is returned. If the operation fails, throws an error.

# duk\_call()

call 1.0.0

## Prototype

```
void duk_call(duk_context *ctx, duk_idx_t nargs);
```

## Stack



## Summary

Call target function `func` with `nargs` arguments (not counting the function itself). The function and its arguments are replaced by a single return value. An error thrown during the function call is not automatically caught.

## Run the function

```
duk_eval_file(ctx, "cool.js");
duk_push_string(ctx, "duktape");

duk_call(ctx, 1);
printf("%s\n", duk_safe_to_string(ctx, -1));
```

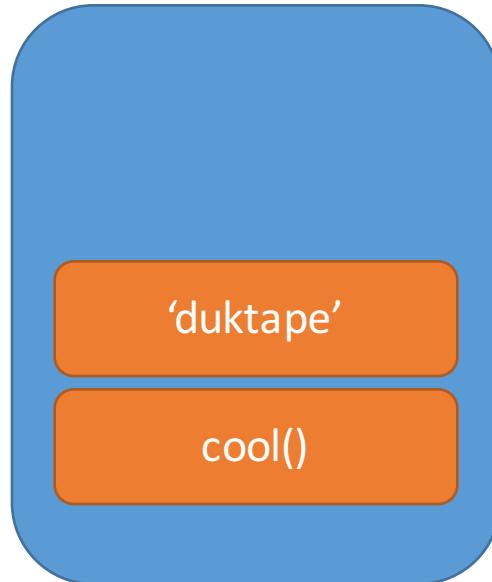
working magic...
duktape is cool

# Stack Attack

eval\_file()



push\_string()



call()



to\_string()

```
int adder(duk_context *ctx) {
    int i;
    int n = duk_get_top(ctx);           // # of arguments
    double res = 0.0;

    for (i = 0; i < n; i++) {
        res += duk_to_number(ctx, i); // add em up
    }

    duk_push_number(ctx, res);
    return 1;                         // single return value
}
```

# Register and Run the C Function

```
duk_push_global_object();
duk_push_c_function(ctx, adder, DUK_VARARGS);
duk_put_prop_string(ctx, 0, "adder");
duk_pop(ctx); /* pop global */

duk_eval_string(ctx, "print('2+3=' + adder(2, 3));");
```

# Return Values

- Return **1** indicates that there is a single return value on top of the value stack
- Return **0** indicates that there is no return value (`undefined`)
- Return **negative** number causes an error to be thrown

# Stack Attack

`push_global()`



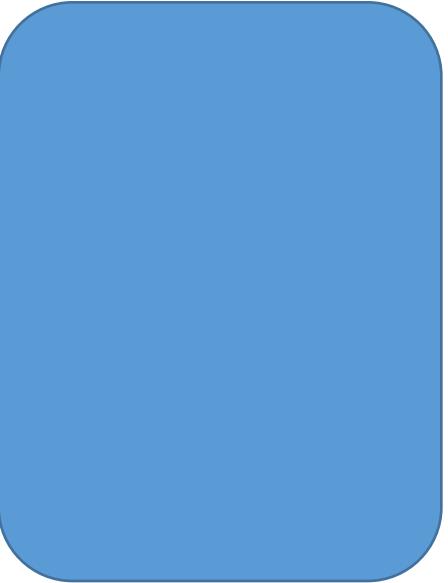
`push_c_func()`



`put_prop()`



`pop()`





Endless possibilities...

- [Dukluv by Creationix](#) (libuv bindings)
- [Dukcurl by Creationix](#) (cURL bindings)