Small, strongly typed, embeddable language. ## Examples

Hello world

```
let {print} = import "std.io"
let world = "world"
print(f"hello {world}!")
```

Async/await

```
let {print} = import "std.io"

let foo = fn()
    print("foo started")
    let bar_frame = async bar()
    print("in foo")
    let bar_res = await bar_frame
    print("foo finished")
    return bar_res

let bar = fn()
    print("bar started")
    suspend
    print("bar resumed")
    suspend
    print("bar finished")
    return 1


print("main started")
let foo_frame = async foo()
print("in main")
let res = await foo_frame
print("main finished:", res)
```

```
$ bog async.bog
main started
foo started
bar started
in foo
bar resumed
in main
bar finished
foo finished
main finished: 1
```

Calculator

```
let {input, print} = import "std.io"

try
    let val1 = input("first argument: ") as num
    let op = input("operation: ")
    let val2 = input("second argument: ") as num

    match op
        "*" => print(val1 * val2)
        "+" => print(val1 + val2)
        "-" => print(val1 - val2)
        "/" => print(val1 / val2)
        "**" => print(val1 ** val2)
        _ => print(f"unknown op: {op}")
catch
    print("that's not a number")
```

Use command line arguments

```
# run with `path/to/bog path/here.bog arg1 arg2 "foo"`
let {print} = import "std.io"
print(import "args")
```

Loops

```
let mut sum = 0
for let c in "hellö wörld"
    match c
        "h" => sum += 1
        "e" => sum += 2
        "l" => sum += 3
        "ö" => sum += 4
        "w" => sum += 5
        "d" => sum += 6

return sum # 31
```

```
let getSome = fn(val) if (val != 0) val - 1

let mut val = 10
while let newVal = getSome(val)
    val = newVal
return val # 0
```

Error handling

```
let {input, print} = import "std.io"

let fails_on_1 = fn(arg) if arg == 1 error(69)
let fails_on_2 = fn(arg) if arg == 2 error(42)
let fails_on_3 = fn(arg) if arg == 3 error(17)

let foo = fn(arg)
    try
        fails_on_1(arg)
        fails_on_2(arg)
        fails_on_3(arg)
    catch let err
        return err

    return 99

print(for let i in 0:4 foo(i)) # [99, 69, 42, 17]
print(try fails_on_1(input("give number: ") as int) catch "gave 1")
```

Destructuring assignment

```
let add = fn ((a,b)) a + b
let tuplify = fn (a,b) (a,b)
return add(tuplify(1,2)) # 3
```

Embed

```
const bog = @import("bog");

var vm = bog.Vm.init(allocator, .{ .import_files = true });
defer vm.deinit();
try vm.addStd();

const res = vm.run(source) catch |e| switch (e) {
    else => |err| return err,
    error.TokenizeError, error.ParseError, error.CompileError, error.
        RuntimeError => {
        try vm.errors.render(source, out_stream);
        return error.RunningBogFailed;
    },
};

const bog_bool = try res.bogToZig(bool, &vm);
```

Calling Bog functions from Zig

```zig
var vm = Vm.init(allocator, .{});
defer vm.deinit();

const res = vm.run(source) catch |e| switch (e) {
    else => |err| return err,
    error.TokenizeError, error.ParseError, error.CompileError, error.
        RuntimeError => {
         try vm.errors.render(source, out_stream);
         return error.RunningBogFailed;
    },
};

const call_res = vm.call(res, "bogFunction", .{1, true}) catch |e|
    switch (e) {
    else => |err| return err,
    error.TokenizeError, error.ParseError, error.CompileError, error.
        RuntimeError => {
         try vm.errors.render(source, out_stream);
         return error.CallingBogFunctionFailed;
    },
};

const bog_integer = try call_res.bogToZig(i64, &vm);
```

Calling Zig functions from Bog

```zig
const my_lib = struct {
    pub fn pow(val: i64) i64 {
        return val * val;
    }
};

var vm = Vm.init(allocator, .{});
defer vm.deinit();
try vm.addPackage("my_lib", my_lib);

const res = vm.run(source) catch |e| switch (e) {
    else => |err| return err,
    error.TokenizeError, error.ParseError, error.CompileError, error.
        RuntimeError => {
         try vm.errors.render(source, out_stream);
         return error.RunningBogFailed;
    },
};

const bog_integer = try res.bogToZig(i64, &vm);
```

```
std.debug.assert(bog_integer == 8);
```

```
let {pow} = import "my_lib"

return 2 * pow(2)
```

Setup

- Download master version of Zig from https://ziglang.org/download/
- Clone this repo
- Build with `zig build`
- Run with `./zig-cache/bin/bog`