Dao

(programming language)

Step aside

- borrows heavily from various languages
- every idea rethought (with substantially different results from the origin)
- e.g.
  - enums (Ruby)
  - ranges (Python)
  - defer (Go)
  - „future“ concurrency (Scala)

Language “market“

- ✔ functional prog. concepts, advanced typing system, immutability etc.
  - expressiveness, readability and safety
  - **required** control (run-time → compile)
- ✔ broadening adoption of scripting languages (increase in code's size)
  - readability and controllability more important than ease of use

Design

- ✔ VM (with GC), interoperability
- ✔ fast exec. in mind and prepared for JIT
- ✔ UTF-8
- ✔ sophisticated type system (inference, compile-time, templates, any, type holders)
- ✔ modularity (OOP, modules, interfaces)
- ✔ control flow

Design details

- ✔ modules (Dao, C, C++)
- ✔ higher-order functions, code sections (with X, Y), code deferring
- ✔ classes, interfaces (abstract, concrete), mixins, wrapped types
- ✔ invar, **casting**, recursive…
- ✔ exception handling, concurrency means

Example 1

```
0$ dao -e 'io.writeln("Hello world!")'
Hello world!
= none
0$ dao -e ' "Hello world!" '
= Hello world!
0$
```

Example 2

```
0$ cat ex2.dao
#!/usr/bin/dao
routine main(... as args) {
  io.writeln(args)
  return 2
}
0$ ./ex2.dao
( )
2$ ./ex2.dao abc 5 7
( "abc", "5", "7" )
2$
```

News 1

- index ranges **[ ]** → **[ )** (like Python)
- **removed** function & class decorators
- **enhanced** concrete interface „replacing“ function & class decorators
- **removed** AOP („replacable“ by mixins, interfaces, …)

News 2

- ✔ **removed** BNF syntax macros
- ✔ **removed** customized verbatim strings (e.g. foreign lang embedding)
- ✔ many **internal** simplifications and optimizations

Interfaces

- any object matching the whole signature of an interface can be **casted** to that interface
- a concrete interface for an interface is similar to a mixin

- casting is good in Dao

Example 3a

```
load time
interface DateTimeX {
  routine add(invar self: time.DateTime,
    ...: tuple<enum<years, months, days>, int> as
vargs) => time.DateTime
}
interface DateTimeX for DateTime {
  routine add(invar self: time.DateTime,
      ...: tuple<enum<years, months, days>, int> as
vargs) => time.DateTime {
    self.add(vargs, ...)
    return self
  }
}
```

Example 3b

```
load web.html import html

io.writeln( document {
  head { meta(http_equiv=$content_type, content='text'
/ 'html; charset=utf-8') }
  body { h1(style='font-size: 50pt') {
    'My page at ' + (string)
((DateTimeX)time.now()).add(years=3)
  } }
})
```

Example 3c

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="content-type" content="text/html;
charset=utf-8">
</head>
<body>
  <h1 style="font-size: 50pt">
    My page at 2019-11-05 10:58:17.254210 +1:00
  </h1>
</body>
</html>
```

Example 4

```
interface Callable { routine ()( ... ); }
interface Callable for routine {
    routine ()( ... as args ){
        io.writeln( "Callable<routine<>>()" );
        self( args, ... )
    }
}
routine Test( par: string ){ io.writeln( "Test", par ) }
const T: Callable = Test;
T( "ABC" );
```

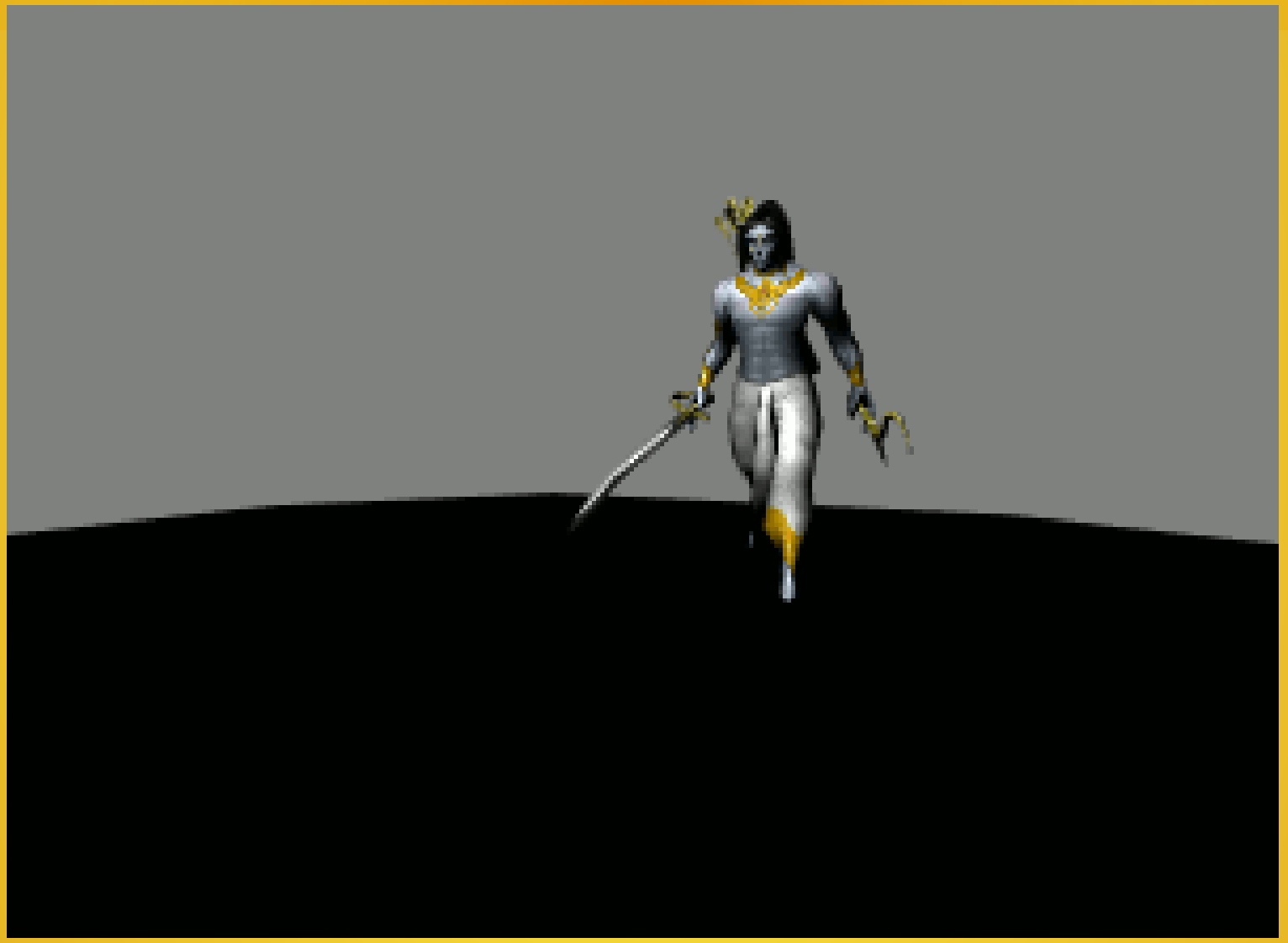
Example 5

```
interface TestUnit { routine Run() }
interface Testing { routine RunTest() }
interface Testing for TestUnit {
    routine RunTest(){
        io.writeln( "Testing<TestUnit>::Run()" )
        self.Run();
    }
}
class OneTest {
    routine Run(){
        io.writeln( "OneTest::Run()" )
    }
}
var test: Testing = OneTest()
test.RunTest()
```

Use

- ✔ bioinformatics
- ✔ computational mathematics
- ✔ parallel and distributed systems
- ✔ „fat“ embedded (e.g. IoT)
- ✔ GUI
- ✔ games
- ✔ **fun!**

Use



Project

- http://daoscript.org (powered by Dao)
- https://github.com/daokoder/dao
- subprojects (dao-modules, JIT, Graphics - 3D, SQL, SDL, CXX, Studio, Emscripten web, ...)
- 6 main contributors, 7 other contributors (data from Nov 2016)
- reference VM (exe 9896 B, lib 926192 B)