Boo!

Scott Stephens

Agenda

- Why Boo?
- Basic Syntax
- Duck Typing
- Meta programming

Why Boo? → C# Pro/Con

- Static typing enables phenomenal intellisense, catches a lot of my mistakes at compile time
- .NET standard library is jam-packed with awesome
- but...
- it's a little wordy
- No interactive console or support for scripts

Why Boo?

- I love Python
- Syntax so crisp, clean, and refreshing
- The console and scripting are ever so convenient.
- but...
- The GIL drives me nuts
- Tight loops can be painfully slow
- I get burned often by typos

Why Boo?

BestOf(C#)

\+ BestOf(Python)

\+ Extra Secret Sauce

Boo

Boo...

- has Python-inspired syntax
- is statically typed
- is type-inferred
- runs on .NET/Mono (libraries, threading, perf)
- Sports an interactive console, an interpreter, and a compiler

but...

- Tool support isn't nearly as good as C#
- Cross-platform isn't nearly as good as Python
- Good luck hiring an experienced Boo developer

Extra Special Sauce

- Optional Dynamic/Duck Typing
- Meta programming facilities

Hello World

```
print "Hello World!"
```

Basic Imperative Syntax

```
a = “Hello world!”
b as int
b = 5
if b == 5:
    print a
else:
    print “Goodbye world :-(“
```

For Loops

```
for ii in range(0,10):
    print ii

for ii in range(10):
    print ii
```

Exceptions

```
try:
    print “Not dangerous”
except:
    print “Just in case”
ensure:
    print “Got to clean up”
```

Simple Class

```
class Adder:
    _addend as int
    def constructor(addend):
        _addend = addend
    def addit(input as int):
        return _addend + input

a = Adder(4)
print a.addit(5)
```

Properties

```
class PropertyExample:

    [Property(Prop1)]
    _field1 as int

    [Getter(Prop2)]
    _field2 as int
```

Properties (the hard way)

```
class PropertyExample2:
    _field as int
    Prop as int:
        get:
            return _field
        set:
            _field = value
```

Generics

```
import System
import System.Collections.Generic

fav_num = Dictionary[of string,int]()
fav_num["scott"] = 7
fav_num["matt"] = 5
```

Slicing

```
num_array =
  ("zero","one","two","three","four")
sub_array = num_array[1:3]
```

Generators

```
num_array =
  ("zero","one","two","three","four")


first_few = (
    x for x in num_array
    if x.CompareTo("p") < 0
)
```

Duck Typing

- The vast majority of code I write behaves in a statically typed fashion (which is why I'm cool with a statically typed language).
- But there are cases where true dynamism is useful.

Simple Duck Typing

```
a as duck
a = “hello”
print a
a = 5
print a
```

Dynamic Arguments

```
def needs_retrieve(r as duck):
    r.Retrieve(“yahoo.com”)
```

Duck Typing

- CSV reader example

Macros

```
print “Hello world!”

lock:
    some_data.Add(“element”)
```

Macros In General

```
macroname <arg1>, <arg2>, … :
    <block of code>
```

Macros

- A macro is a parameterized transformation of code.
- The macro arguments are the parameters of the transformation.

Macros

- Repeat It example
- Timer example

You may want...

- A copy of this presentation:
  - github.com/scottstephens
- Boo Downloads and Documentation
  - boo.codehaus.org
  - github.com/bamboo