Gotchas for Python Users

Learning boo if you are familiar with Python

It might come as no surprise to anyone but I'd better state it here anyway: **boo is not python**.

It looks a lot like python and there's a reason for that: **we love python!** Specially *The dirty hungarian phrasebook* 🙂

General Tips

Learn about .NET & Mono

The .NET Framework (Windows) or Mono (Linux,Windows,Mac) is required to run or compile boo programs.

.NET Assemblies

Instead of using the python standard library and python modules, you will be using compiled assemblies. You have access to the standard class library that is included with .NET and Mono. There are many 3rd party .NET libraries available as well. See the General Help page for a list of some, as well as pointers to more .NET information.

GUI Toolkits

For GUI programming the two predominant options are System.Windows.Forms and GTK#. See the General Help p age for links to tutorials for learning how to use those GUI toolkits.

General things in common with Python

Syntax-wise, boo is very similar to python. Like python, you indent code blocks instead of using curly braces {}. Methods look like python methods (except for the optional static type declarations):

```
def mymethod(s as string):
    print s.ToUpper()

mymethod("a sample string")
```

See the Language Guide for info on lists, dictionaries, classes, for loops, generators, list comprehensions, etc.

Boo also has an Interactive Interpreter like Python.

Duck Typing vs. Static Typing

This is the major functional difference between boo and python (other than the fact that boo runs on .NET and Mono).

See Duck Typing for a more in-depth explanation. In Python, everything is duck typed implicitly. Type-specific operations are not resolved until run-time. In boo, everything is implicitly static typed at compile time. You often do

not have to explicitly declare the type, due to boo's Type Inference mechanism. For example, you can still simply state "x = 4" instead of having to state "x as int = 4".

You can however specify that an object is to be duck typed by explicitly declaring its type as "duck". Also, boo now has an option that turns on implicit duck typing by default, which makes coding in boo much more like python. Again, see Duck Typing for more info.

Specific Differences Between Python & Boo

Print

Print can be called as a function or as a statement:

```
print("is now a function")
print "and a macro statement"
#or you can use
System.Console.WriteLine("...")
```

See Syntactic Macros for other useful macro statements in boo such as assert, debug, using, lock, with, and performTransaction.

Embedding variables in a string

Use a $ sign followed by the variable name or $ sign followed by the expression enclosed in parentheses:

```
"first letter is: $firstLetter"

"x + y = $(x + y)"
```

You can triple quote strings too just like python.

See String Operations for more info.

Importing

Importing automatically puts everything into the namespace, just like C#. So when you import an assembly, it is like saying "from MyLibrary import *" in Python.

```
import MyLibrary
print Version #same as print
MyLibrary.Version
```

If you import two or more libraries that have the same named item, the compiler will tell you, and you can use the fully qualified name ("MyLibrary.Version" instead of just "Version").

Commenting

Boo supports Python-style commenting, but also C++ commenting:

```
#commented line
//commented line
/*
multiple lines
commented out
*/
```

Lists, Arrays, Slicing, Generators

You can do all the same list operations, list comprehensions, slicing, and generators as in python. Instead of tuples however, there are C-like arrays - fixed length and of a particular type.

See Lists And Arrays, Slicing, and Generators for more information.

Dictionaries / Hashtables

In boo, {} is a Boo.Lang.Hash class, which subclasses System.Collections.Hashtable, but it works similar to Python dictionaries.

See Hashtables.

No "self" required!

In Python, you use "self" to signify instance variables and instance methods of classes:

```
self.x
def mymethod(self):
```

In boo, you can just use:

```
x or self.x
def mymethod():
```

You do not need to declare "self" as the first parameter of an instance method of a class. And you can still use "self.x" but the self is optional. In boo, "self" basically means the same as "this" in C#.

*_init, __del_* (Constructors & Destructors)

Instead of *_init_*(self), define a method named "constructor()" in your class. "destructor()" is used for destructors.

Main & argv (command line arguments)

Instead of if name == *_main_*:

"argv" is passed to your script as an array of strings. See examples/download.boo, or you can call Environment.GetCommandLineArgs().

Note though, the "main" (or global) part of your script that is executed when run has to be at the bottom of the script, below any import statements, classes, or def functions. See Structure of a Boo Script.

Documentation strings

In Boo, docstrings must start at the same indentation level as the class/method/function/callable definition, and you must use triple quoted strings:

```
class MyClass:
""" MyClass' docstring """
    def foo():
    """ foo's docstring """
        pass
```

There is an advantage to doing it this way. In Boo, we can use docstrings for anything, including properties, fields, namespaces, and modules, as well as classes and functions. See tests/testcases/parser/docstrings_1.boo for a more complete example.

We can convert these docstrings into an XML format which the NDoc tool can then use to generate nicely formatted HTML documentation for your code. Or you can try the Monodoc approach instead. You pass it your compiled exe or dll, and write your user documentation externally and separately from your code. This lets you focus on making your code as simple and readable as possible, and not over-cluttering it with docstrings.

true and false

Booleans true and false are not capitalized like in Python. They are lowercase like in C#, javascript, java, and other languages.

char vs. string

In Python, there really is no special "char" type, like in .NET. You just use strings with a single character, like 't'.

Since Boo utilizes the .NET Class Library and is statically typed, it needs that distinction, however. char('t') refers to a System.Char type, whereas "t" or 't' is a System.String.

Static vs. Instance fields and methods

In Python, you might refer to a static field shared by all class instances using notation like "MyClass.y". It is the same in boo.

To create a static field or method like this that is shared by all class instances, use the static keyword: "static public y" or "static def mymethod():".

NOTE Any fields or constants you declare in a class are by default "protected" and not accessible from outside the class (methods or properties are by default public). Add a "public" modifier in front of the field to make it accessible, or else create a property instead (with a getter and setter, see example in later section).

Named parameters, Assignment Expressions, Set Properties via Constructor

Boo doesn't support named parameters. It does however support setting properties via the call to the constructor of a class.

For example this code:

```
pt = Point(X:100, Y:200)
```

is essentially the same as this:

```
pt = Point()
pt.X = 100
pt.Y = 100
```

Thus X and Y must refer to public fields or properties, not private or protected fields. You can also see see why the

constructor in the Point class doesn't need to handle the X and Y parameters itself. In fact your Point class doesn't even need a constructor. If your class doesn't have a constructor, Boo will generate it for you.

Why a colon (:) instead of an equal sign (=)? Because in boo assignments (like x=100) are expressions that are evaluated, not just statements like in python. If you pass "x=100" to a method, for example, it will essentially pass the value 100, after assigning 100 to the variable x.

Boo supports handling a variable number of parameters using the same syntax as python (*params).

*_str_* (String representation)

Instead of *_str_* or *_repr_*, define a ToString() method in your class.

Overloading operators: *_add, __mult_*

See Operator overloading.

*_call_* (Callable)

In Python, you might override the *_call_* method in a class to make it callable.

In boo, check out options like Callable Types, the ICallable interface, Events, and anonymous Closures.

__ (double underscore for private variables)

Use the private modifier: "private x as int"

Properties (with Getter, Setter)

Instead of x = property(getter,setter)
use syntax like the below:

```
class MyClass:
    _fld = 3
    MyProperty as int:
        get:
            return _fld
        set:
            _fld = value
```

or you quickly create a property with the default getter and setter like so:

```
import System.Drawing

class MyClass:
   [Property(Font)] _font as
System.Drawing.Font
```

See the .NET docs on properties for more info.

Decorators/Attributes

Instead of

```
@decorator
def mymethod():
```

boo has attributes (see .NET's docs on attributes).

```
[attribute]
def mymethod():
```

_name, __class, __file_

See the .NET documentation under System.Reflection on Types and GetType. You can inspect a class for example like this:

```
y = MyClass()
print y.GetType().Name
for m in y.GetType().GetMembers():
 print(m) //or m.Name for just the short
name
```

Instead of *_file_*, there are related options like:

```
Assembly.GetExecutingAssembly().Location
//or
System.AppDomain.CurrentDomain.BaseDirectory
```

Pickle (saving and loading objects)

See XML Serialization, especially the bottom example showing how to save and load a dictionary/hashtable.

Things in Python But Not Boo

Importing and dynamic importing (*_import_*) of other python files

Not available in boo, but see examples/pipeline/AutoImport. You can only import compiled assemblies. The autoimport example compiles the other boo script to an assembly and then imports it.

Note: To combine multiple boo scripts into one boo application, instead of importing one script from another script like in python, you would pass *all* your boo scripts to the compiler (booc) at the same time. See How To Compile boo scripts.

When creating more complex applications in boo that require multiple scripts, I recommend using the SharpDevelop IDE with the boo add-in on Windows, or else Nant, a .NET build tool similar to java's ant. On Linux, there is now a boo add-in for the MonoDevelop IDE.

Dynamically modify class methods on the fly after creation

The default classes in boo are statically typed. You cannot change or add methods to a class instance at runtime.

But using duck typing there are ways to simulate the dynamic behavior of python classes. See the examples below, especially the 2nd and 3rd ones.

- basic duck type example
- Python-like class using IQuackFu
- Dynamic XML object example using IQuackFu
- Dynamic Inheritance - fun with IQuackFu
- System.Remoting, an alternative to IQuackFu in some cases: RealProxy example

Things in Boo but Not Python

- quickly compile boo script to standalone cross-platform exe
- easy super(), and constructor automatically calls super() for you. If you don't have a constructor, one is created for you.
- set class properties via the constructor (constructor doesn't have to handle them explicitly):

```
x = MyClass(Property1:"value1", Property2:"value2")
```

- Anonymous Closures - including multi-line closures:

```
b = Button(Text:"Press Me")
b.Click += def():
    MessageBox.Show("you clicked me")
```

- Events, Callable Types
- unless statement: print "good job" unless score < 75
- built-in support for Regular Expressions
- timespan literals (example: t = 10ms)
- extensible compiler pipeline
- custom Syntactic Macros

since boo is statically typed, you get:

- static typing: "x as int" but you can just say "x" (x is an object)
- Interfaces, Enums
- private,public,protected,final,etc. variables & methods
- speed increases - without having to convert your code to a different language like C
- easier interoperability since boo uses standard CLI types (e.g. string is System.String, int is a System.Int32...)
- convert C# and VB.NET to boo code (part of the Boo AddIn For SharpDevelop)

other C#/.NET features you get:

- "lock" (like java's synchronized). See the lock* examples under tests/testcases/semantics/.
- property getters and setters
- using: (automatically disposes of object when you are done using it)

```
//in this example you can also use the simpler
//   "for line in StreamReader("filename"):"
using f = System.IO.StreamReader('using0.boo'):
    while line = f.ReadLine():
        print line
```

- parameter checking

```
def foo([required(value > 3)] value as int):
     pass
```

- [attributes] for functions, fields, classes...
- asynchronous execution, see Asynchronous Design Pattern.
- XML Serialization