Testing the performance of a few languages to get a feel for dynamic versus static typed performance. This is to assess whether to make Pie a dynamically typed language, a hybrid, or purely statically typed. In other words, I want to assess if this would even be worth it, before assessing how difficult it would be.

Two benchmarks:

The first simply counts to 1,000,000. It was originally an iterative fibonacci algorithm, but it generated such large numbers that it created integer overflows that may impact performance, so I made it just count numbers in a for loop instead.

The second is a brute-force recursive fibonacci algorithms: I wanted to test recursion because, thanks to late binding, recursion is where dynamically typed languages tend to fall on their faces the most.

Times are in seconds.

Language	Count to n = 1000000	Recursive Fib to n = 30
C++	Too fast to measure!	0.003003400
C# with static typing	0.0025659	0.0097299
C# with dynamic typing	0.131144	0.5136198
IronPython	0.1320776	0.1834815
Python	2.3186409	13.61336

As we can see, C++ is the clear winner for raw performance. No shock there, native code will always beat anything else when testing algorithm performnce. In fact, the counting test ran too quickly for even a high precision timer to time accurately.

C# static typed code beats the dynamic languages handily. Also no big surprise.

C# has built in support for dynamic typing (I love this language!). When using dynamic typing, it matches IronPython and trounces Python for the iterative counting test. However, IronPython beats dynamic C# for the recursive test.

IronPython (a .NET version of Python) matches or beats C# dynamically typed performance. I'm rather surprised at the performance of IronPython with recursion: they must have done something fancy to manage that.

Lastly, Python performs abysmally in both cases.

Summary: My first thought was that adding dynamic typing support to Pie would be silly when you can use IronPython. However, IronPython hasn't been updated in three years: that does not fill me with confidence. So, I'll proceed with proof of concept code to test viability.