Progress Report

To build the boa interpreter, run "make", pertaining to the makefile in the src directory. In order to run a boa program that makes use of classes and objects, you will have to run the command `boa [filename] -nocheck`.

Boa programs can be written in a similar manner to Python programs, with a couple of key differences. - Function definitions must begin with a "def" keyword, followed by the name of the function, and a parenthesized, comma-separated list of arguments. After the argument list, one can optionally include an arrow `->` pointing to the return type of the function, followed by a newline and a block of code (if this is omitted, the type will be inferred). - Class declarations require a list of all data members at the beginning of the declaration. These data member declarations are variable declarations without an assignment, preceded by the keyword "member" (this will probably be removed in the future). Instance methods must have at least one parameter, which will be passed the object instance upon a method call. Instance initializers are just as they are in Python - there can be a single initializer in a class, and it must be named `__init__`. Currently, `__init__` methods must end with an expression returning "self" (or whatever the first method argument is). Following is the general structure of a class declaration: - ``` class [name]: member let [id1] : [type] member var [id2] : [type]

```
    def __init__(self, [...args]):
      self.[id1] = [expr]
      self.[id2] = [expr]

      return self
```

In this sprint, we really got a lot of the key features we were hoping for in boa implemented. Following is a list of the major goals that we achieved:

- Single-line comments, comprised of `#` followed by text and a newline
- Parsing of indented blocks through whitespace as opposed to curly braces
- Hindley-Milner type inference (Algorithm W)
    - We can currently type-check (with inference) all language constructs, aside from classes
- Static type-checking with parametric polymorphism of programs with optional type hints
- Partial application of both lambdas and functions defined with the `def` keyword
- Mutability enforced in the type system
    - Immutable variables are declared with `let [id] = [expr]`, and mutable variables are declared with `var [id] = [expr]`
    - Variables can be introduced/declared without assigning them a value, and their type will be inferred later:
        - `let x`
        - `x = 500`
    - Immutable variables can only be given a value a single time, but you can re-introduce a variable with the same name as an already-defined variable, with another instance of one of the "var" or "let" keywords:
        - `let x = 500`
        - `# x = 10 would not be allowed, x is immutable`
        - `var x = 10`
        - `x = 100`
        - `# this is now allowed, because x has been re-introduced as mutable`

Self-Evaluation

During this sprint, we had a slight change of direction from what we were planning after beta - we did not get around to implementing list slicing, dictionary accesses, tuple slicing, or the compatibility libary, but we added a new goal - implementing parametric polymorphism with Hindley-Milner type inference. This proved a very challenging and exciting exercise, and we were super excited when we saw that completely type-annotation free boa programs were able to be statically typed. Though our goals changed from the previous sprint, we would give ourselves "good/excellent" scope.

Next Steps

We plan to continue with boa, and hope to fully realize the vision we set out with at the beginning - a language with some of the expressive advantages of Python, but one that allows programmers to also take advantage of static analysis of their programs. Our immediate next steps include the addition of structural typing and type-checking classes, and implementing an object model in order to allow for the evaluation of expressions like list accesses and slicing, operator overloading, and string representation of objects. After that, we hope add both a python compatiblity layer, dependent types, and make a boa compiler.