1 *

2 Introduction: What, Why, and How?

2.1 What is \English?

2.2 What makes it different?

2.3 What is good about these differences?

2.4 How are these differences possible?

3 How to Use the Language

3.1 Compilation

3.2 The Basic Basics

3.2.1 Comments

I mention comments first because they'll be used in future examples within this documentation. The octothorpe ('#') begins single-line comments:

```
some code # a comment
####### comment comment comment
more code ## more comment with more #s!
```

Octothorpes can be paired with angle brackets ('<' and '>') to make multi-line or inline comments:

```
some #< awesome ># code
###< You can have as many #s as you want and they don't
even have to match the number at the end!  >#

code
#< You can also have nested comments to make commenting out sections of code easier.
I commented out the code below even though it also has multi-line comments.

#< This is a comment ># Here is some code that is commented and bad

Still, the comment doesn't end at the first ">" followed by an "#"; it matches them like parentheses.
>#
more code
##< below is a pretty cool operator that we'll explain and even show you how to make later;
this two-# could be used as a doc comment (maybe?)  >#
define <type of chars> string:
    chars;
```

3.2.2 `main(`... Oh, Wait

There is no main method/function/operator. Just write what you want to do!

3.2.3 Variables and Math

\English is strongly, statically typed. Variables need to be declared and need to be declared with a type. Variable declarations follow the same classic C-like format of a type followed by a name. Note that \English convention is the reverse of Java and many other languages: capitalize the first letter(s) of the variable names and keep types lowercase.

```
int Integer
queue Queue
```

In \English, there's a bit of a twist: in this language, types, variable names, and other operators can have spaces, can be defined with special character-matching operators, and more. (See Section 3.4 later on.)

```
int Number of Variables
list of queues of pairs of ints and floats My Beautiful Queues
```

3.2.3.1 Standard Number Types

\English includes a wide variety of different types of numbers; some have static memory sizes and some don't. Here are some of the most basic number types that do *not* have set memory sizes:

- `number` is any kind of number, floating point or integer, static or arbitrary size, etc.
- `decimal` is any kind of rational number, including integers, floating-point numbers, or any special kinds of numbers like integral fractions or square roots calculated lazily. All `decimal`s are types of `number`s.
- `integer` is any kind of integer or any size, arbitrary or static. All `integer`s are types of `decimal`s.

\English also includes a few types of numbers of static size:

- `byte` is an 8-bit integer.
- `int` is a 32-bit integer.
- `float` is a 32-bit (single-precision) floating point number (IEEE-style).

Other features of number types will be described in 3.5.3

3.2.3.2 Variable Declaration and Assignment

3.2.3.3 Standard Arithmatic Operators

3.2.4 Lists and More

3.2.4.1 Making a List

3.2.4.2 Manipulating Lists By Index(es)

3.2.4.3 Strings Are Lists And More!

3.3 Code Blocks

3.3.1 If and Its Friends

3.3.2 Loops

3.4 Operators

3.4.1 Defining Them

3.4.2 Using Them

3.5 Types

3.5.1 Defining Them

3.5.2 Using Them

3.5.3 Type Modifiers

3.5.3.1 Defining Them

3.5.3.2 Using Them

3.5.4 Units

3.5.4.1 Defining Them

3.5.4.2 Using Them

3.6 Errors

3.6.1 Defining Them

3.6.2 Using Them

3.7 Input and Output

3.7.1 Stdin Reading and Stdout Printing

3.7.2 File Reading and Writing

3.8 Implicit Conversion

3.8.1 What?

3.8.2 Why?

3.8.3 Super-implicit?

3.9 High Order Functionality

3.9.1 What?

3.9.2 Why?

3.9.3 Missing Something? Partially Applied Operators

3.9.4 Operator Modifiers

3.9.5 Scope and Contextual Operators

4 The Basis of the Language