AtoZ

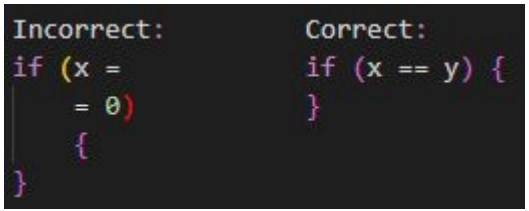
Anthony Bisgood and Zach Empkey

Overview

- AtoZ is a Java-like, statically typed language that is interpreted using Python. Our language implements Java like features including semicolons to determine end-of-lines and brackets for loops and if statements.
- However, our language does take different paths with keywords, declarations, and implementations of features.
- This presentation will describe and explain what and why we decided on design choices, as well as examples of how to write your own program in our language and how to test it.

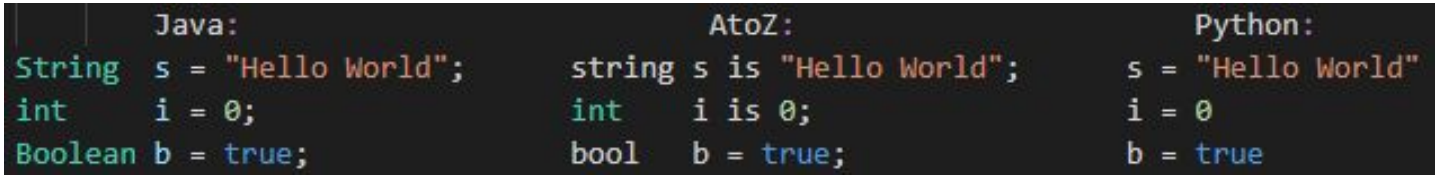
Design - Basics

- AtoZ is a basic language interpreted by Python.
- With statements ending in semicolons and blocks of code within if statements and while loops bordered by curly braces, the syntax is similar to that of C or Java.
- In order to ensure better readability, spaces between certain parts of statements are required. An example is shown later on in the slides.
- Indentation does not matter in our language, however, variable declarations, print statements, and if/while statements must be written on the same line. Ex.



Design - Declarations and Types

- When thinking about how we wanted implement declarations in AtoZ, we wanted something that represented what users were familiar with (Type Name = Value) but add something different that would differentiate ourselves from other languages. We eventually settled on the Syntax (Type Name is Value).
- Type names are string, int, bool, all of which are lowercase. Variable names can be any collection of alphabetical characters.
- For this project we decided to focus on the three main types, int, string, and boolean. We believed that these 3 types would give our language the largest amount of flexibility while still being simple.



Design - Printing and Comments

- While designing our print function, we decided on the keyword show(). We didn't want the wordiness of Java, but still wanted to distance ourselves from Python. When printing, the user can decide to print either a string or a variable that they have saved.

```
Java:                                AtoZ:                              Python:
String s = "Hello World";            string s is "Hello World";         s = "Hello World"
System.out.println(s);               show(s);                           print(s)
System.out.println("Hello World");  show("Hello World");               print("Hello World")
```

- In AtoZ, comments are denoted by the character "@". Lines that start with @ are ignored.

```
Java:          AtoZ:          Python:
//Comment      @Comment       #Comment
```

Design - Command line arguments

- AtoZ handles command line arguments simply with the arg() macro.
- The user is required to give both the type and the value.
- If you want to take a command line argument, simply write:

  arg(X);

  In your script, and then run from the command line with:

  python3 language_translator.py file_name.txt <type> <value>

Design - Logical Operators

- Logical Operators in A to Z include can be used over all 3 types in the language, including variables of the same type, and can be used between variables and primitives.

```
string s is "fwa";
if (s == "fwa") {
    doSomthing;
}
```

English	AtoZ
Greater Than	>
Greater Than Equal To	>=
Less Than	<
Less Than Equal To	<=
Equals To	==
Not	!
And	&&
Or	||

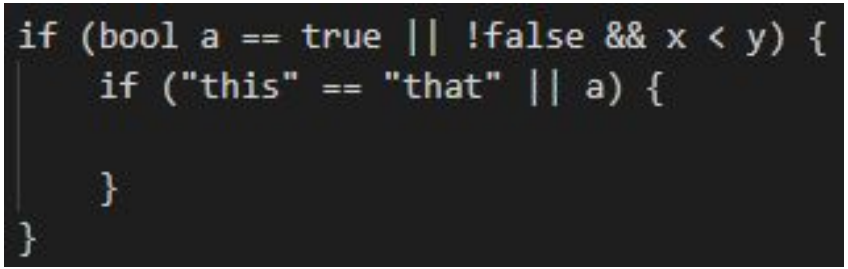
Design - Mathematical Operators

```
int y is 14;
int x is 32;
int x is y + x;
```

- Mathematical Operators in AtoZ include: Addition, Subtraction, Division, Multiplication, and Modulation.
- Like logical operations, mathematical operations can be used between declared variables and primitive types.
- In addition, when wanting to change a variable that you have declared, you MUST DECLARE THE TYPE of the variable you wish to change.
- All operations over integers that result in a decimal are floored to the next lowest integer. (ex. int x is 5/2; the result is x == 2)

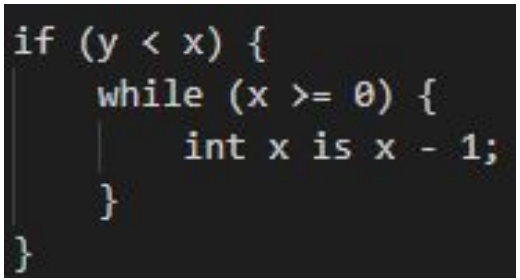
Design - If Statements

- If statements begin with the word “if”, followed by () { and a closing bracket on its own line }.
- In addition, the conditional body of an if statement () must be followed by a space then an opening bracket on the same line {. After the “(“ in body, there must be no space between it and the condition. So if (x==1) works but not if (x == 1).
- The conditional body of an if statement must be a boolean expression that either evaluates to true or false. The body can contain multiple statements and include boolean, string, or integer comparison. However, you may not compare 2 values/variables of different types.



Design - While loops

- While loops in AtoZ function the same way as they do in other programming languages like Java or Python. In the body of a while loop is a conditional statement, and if it is true, then continue to iterate through the loop, and if false, break the loop and move to after the loop.
- While loops can be nested, and may include other if statements and declarations. However, any variable declarations in a while loop are scoped globally. So a change inside a while loop to a variable outside the loop will change the outside variable.
- The restriction of brackets in if statements applies to while loops.



Restrictions

- Cannot do mathematical or logical operations between 2 variables of a different type
- Cannot perform operations inside of show(), you must store the output of your operation into a variable, then pass show() that variable.
- Cannot print a string that is the same as a declared variable.
    - Ex string s = “hello s ”
        - If (s == “hello s “)
    - This would cause an Error

Error Handling

- Our language implements a handful of Error Handling methods. This includes type checking for declarations, operations, as well as syntax handling. In addition, if the user writes code that is incorrect, an error message will print indicating the line and variable that was wrong.

Difficulties

- Initially getting the regex dialed in was quite difficult, and required many attempts. But once the first few statements were written, the process went much smoother.
- Getting the while loops to run properly through the interpreter was a challenge. We needed a way to ensure that statements would run cyclically as long as the loop condition evaluated to true. To do this, upon seeing a while statement, the interpreter would create an array and populate it with the condition and subsequent statements until it reached its respective end curly brace. After which it would call a helper function that would call the main parser function and pass in the statements from the while block as though they were their own file. This semi recursive solution also allows for nested loops.

What's Next

- If we were to expand the language further, we would definitely implement some sort of data structure to help store and retrieve data.
- In addition, adding more types such as double, and adding an implementation for "for" loops is one of the top things on the priority list.
- Furthermore, adding better handling and letting the user be more free with their typing, e.g if/while loop () restrictions and line restrictions is important.
- Lastly, we would like to add scoping, and eventually functions/methods, or even classes to the language. This would significantly increase the usability of the language.