“Never waste any time you could spend sleeping” by someone
Project Manager
Lennart Hardenberg

System Integrator
Kofi Boateng

System Architect
Shangshang Chen

Language Guru
Eric Maciel

Testing Ninja
Max Weber
What is Wikify?

Efficient

Programming language
data processing
to learn
Motivation

Frustrated how hard it was to work with data from Wikipedia

We see lots of potential for journalists and data hackers to make use of it in the future.
Project Management

400 Emails sent = 1/day/person

- Github
- Google Docs & Google Slides
Types

- num
- string
- bool
- page
- table
- image
Newline Terminated Language
Keywords:
- for, while, break
- if, else
- func
- end
func println( string line )
    print(line + "\n")
end
Syntactic Constructs

if (cond)
    ...
else
    --> optional
    ...
end

if (cond)
    ...
end
prog
  import_seq prog_seq
  seq
    main func
      main ()
      stmt_seq
      end
    stmt
      print ( expr )
        str_expr
          "hello, world\n"
Enough Talk, Show Me Some Wikify
main()
    page p1
    p1.urlPrompt()
    print(p1.getUrl())
    p1.returnInfobox()
end
Example 2 - Table to Excel

main()
    page p1
    p1.urlPrompt()
    print(p1.getUrl())
    //put things into an excel sheet
table t1
    t1.url(p1.getUrl())
t1.getTable(0)
t1.toExcel("file")
end
Translator architecture

WikiLexer → Tokens → WikiParser → AST → Code Generator

main()
print("Hello, World\n")
end

[@0,0:3='main',<44>,1:0]
[@1,4:5=(''),<2>,1:4]
[@2,6:6='\n',<51>,1:6]
[@3,8:12='print',<35>,2:1]
[@4,13:13=(''),<3>,2:6]
[@5,14:22='"hello\n"',<50>,2:7]
[@6,23:23='(',<4>,2:16]
[@7,24:24='\n',<51>,2:17]
[@8,25:27='end',<37>,3:0]
[@9,28:28='\n',<51>,3:3]
[@10,29:28='EOF',<-1>,4:0]
Translator architecture

java code

Java Virtual Machine

Wikify libraries

Page.java
Table.java
Image.java

apache.poi
jexcel library
jsoup library

“Hello, World”
Hello, World
**Software development environment**

**SDE for translator:**
ANTLR

(ANother Tool for Language Recognition)

**Lexical analysis, automatic parse tree generation:**

Grouping input characters into IDs, nums, NL, WS, comments, strings,....

The tokens consist of two pieces of information, the token type (which identifies the lexical structure), and the text matched by that token by the lexer.

**ANTLR V4 produces recursive descent parse trees:**

This means that sometimes, the parser needs a lot of lookahead tokens to know which grammar production to expand. ANTLR deals with that.
Resolving Ambiguities

Resolving Ambiguities:

The ANTLR parser chooses the first production specified (when it sees an ambiguous phrase) ANTLR also matches the input string to the production specified first in the lexer to resolve ambiguities
Parse Tree Listeners and visitors

Listeners and Visitors
ANTLR V4 automatically generates a tree listener to listen and react to triggered events.

Each node in the generated parse tree has an enter() and exit method

On the enter and exit methods for each grammar production we output Java code to a buffer which is then output to a file
Testing

Unit tests for each developing phase
   → Pass and Fail test cases
   → Using Wikipedia input for testing

Regression Testing
Testing of programs in the wiki library
   → HtmlParser

Problems detected
   → empty function did not translate properly
   → Function efficiency
“Never waste any time you could spend sleeping”

...developing your next cool programming language