“Never waste any time you could spend sleeping”

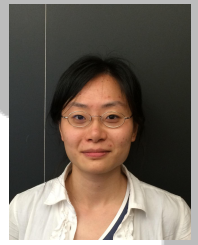
by someone

TEAM TEAM TEAM

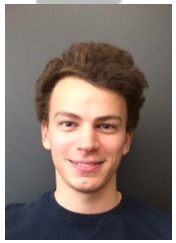
Project Manager
Lennart Hardenberg

Language Guru
Eric Maciel

System Integrator
Kofi Boateng

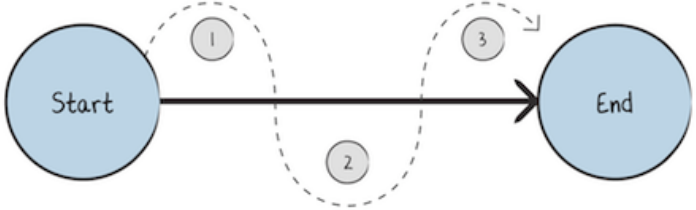


System Architect
Shangshang Chen



Testing Ninja
Max Weber

What is Wikify?



Efficient

to learn

Programming language

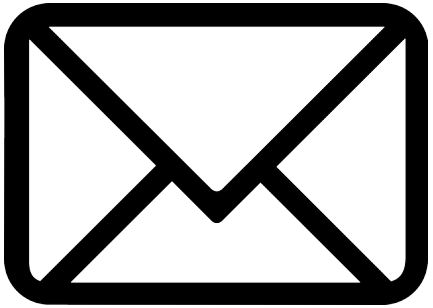
data processing

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

Syntactic Constructs

Newline Terminated Language

Keywords:

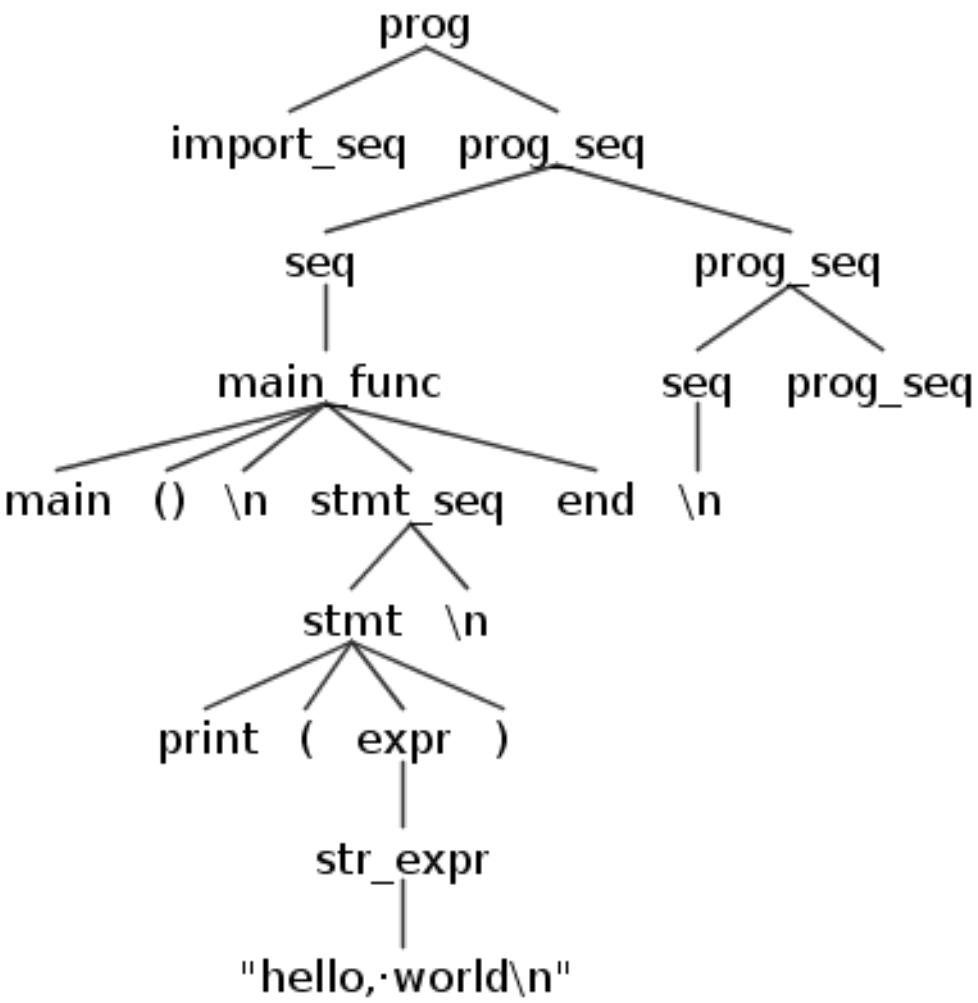
- for, while, break
- if, else
- func
- end

Syntactic Constructs

```
func println( string line )
        print(line + “\n”)
end
```

Syntactic Constructs

```
if (cond)                                   if (cond)
        …                                           …
else  ------>    optional          end
        …
end
```


prog
import_seq
prog_seq
seq
prog_seq
main_func
seq
prog_seq
main
()
\n
stmt_seq
end
\n
stmt
\n
print
(
expr
)
str_expr
"hello,·world\n"

Enough Talk, Show Me Some Wikify

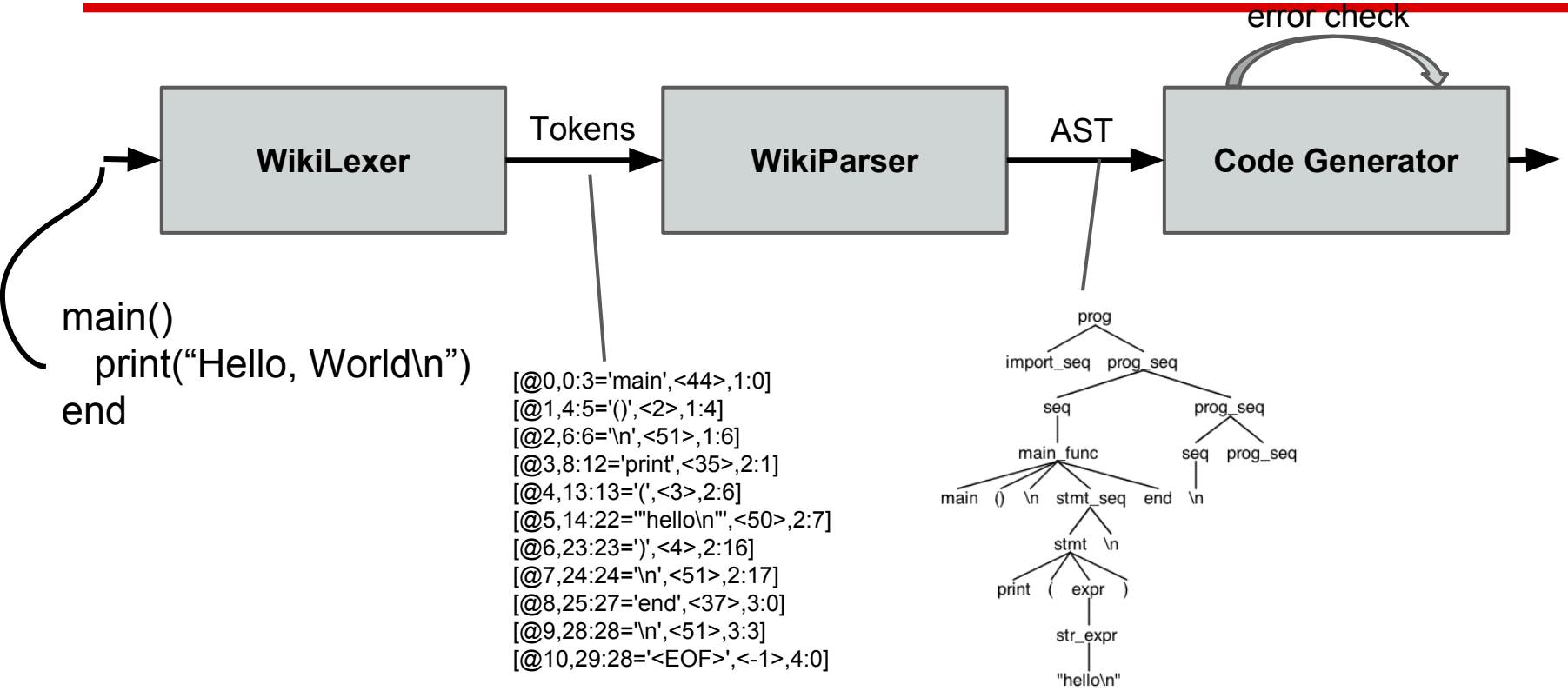
Example 1 - Infobox Example

```
main()
        page p1
        p1.urlPrompt()
        print(p1.getUrl())
        p1.returnInfobox()
end
```

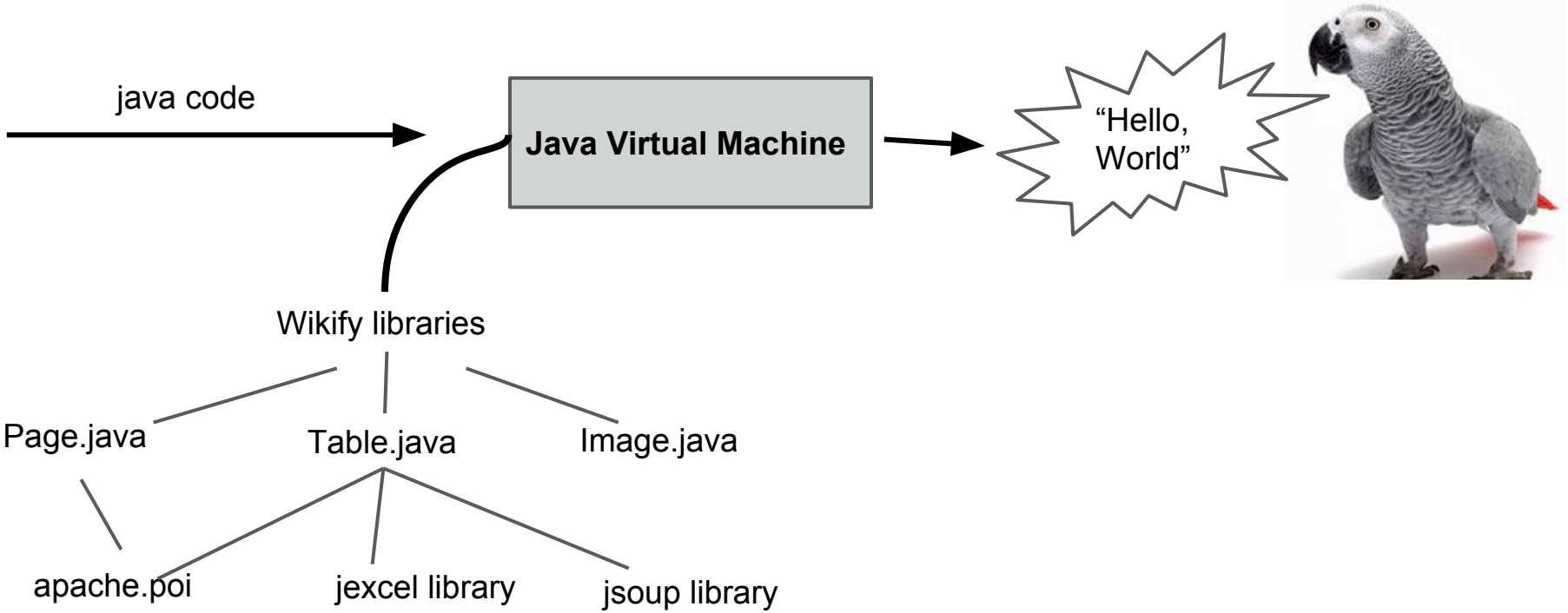
Example 2 - Table to Excel

```
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



Translator architecture



Runtime Environment

- cd WikiLang
- ./wikit file.wiki

========Translation Successful=======

- ./wiki file
- Hello, World

Software development environment

SDE for translator:

ANTLR

(ANother Tool for Language Recognition)

Lexical analysis, automatic parse tree generation:

Grouping input characters int ID's, nums, NL, WS, comments, strings,....

The tokens consists 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

Conclusion

"Never waste any time you could spend sleeping"

...developing your next cool programming language