Parsing Languages with mxTextTools

Building fast compilers in Python

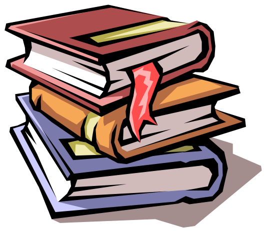
EuroPython Conference 2007
Vilnius, Lithuania

Marc-André Lemburg

EGENIX.COM Software GmbH
Germany

Speaker Introduction: Marc-André Lemburg

- CEO eGenix.com and Consultant
  - More than 20 years software development experience
  - Diploma in Mathematics
  - Expert in Python, Application Design, Web Technologies and Unicode
  - Python Core Developer
  - Python Software Foundation Board Member (2002-2004)
  - Contact: mal@egenix.com
- eGenix.com Software GmbH, Germany
  - Founded in 2000
  - Core business:
    - Consulting: helping companies write successful Python software
    - Product design: professional quality Python/Zope developer tools (mxODBC, mxDateTime, mxTextTools, etc.)
  - International customer base



Agenda

1. Introduction to mxTextTools
2. Parsing with the Tagging Engine
3. Compiling Languages with mxTextTools
4. Discussion

Introduction to mxTextTools

mxTextTools: Motivation

- Regular Expressions are sometimes nice…

  r′ \d\d\d\d-\d\d-\d\d′

  but often incomprehensible:

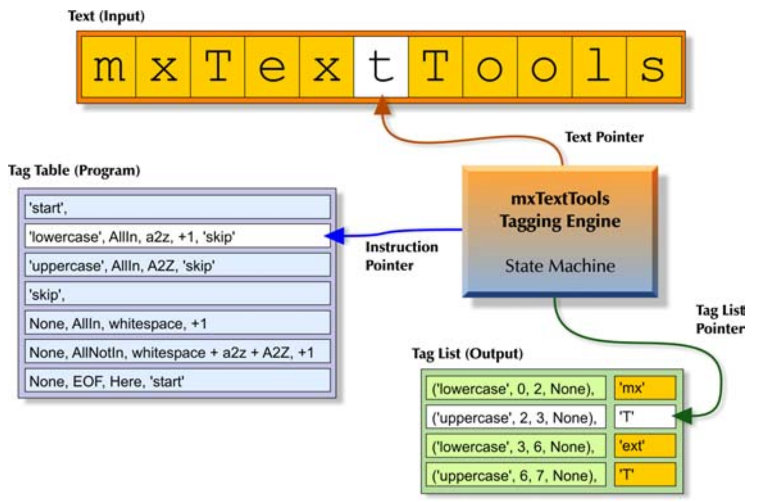
  ```
  (r′\s*([a-zA-Z_][-:.a-zA-Z_0-9]*)(\s*=\s*′
   r′(\'[^\']*\'|"[^"]*"|[]′
   r′[\-a-zA-Z0-9./,:;+*%?!&$\(\)_#=~\'"@]*))?′)
  ```

  (this parses an SGML attribute)

- Better use the good old iterative approach…

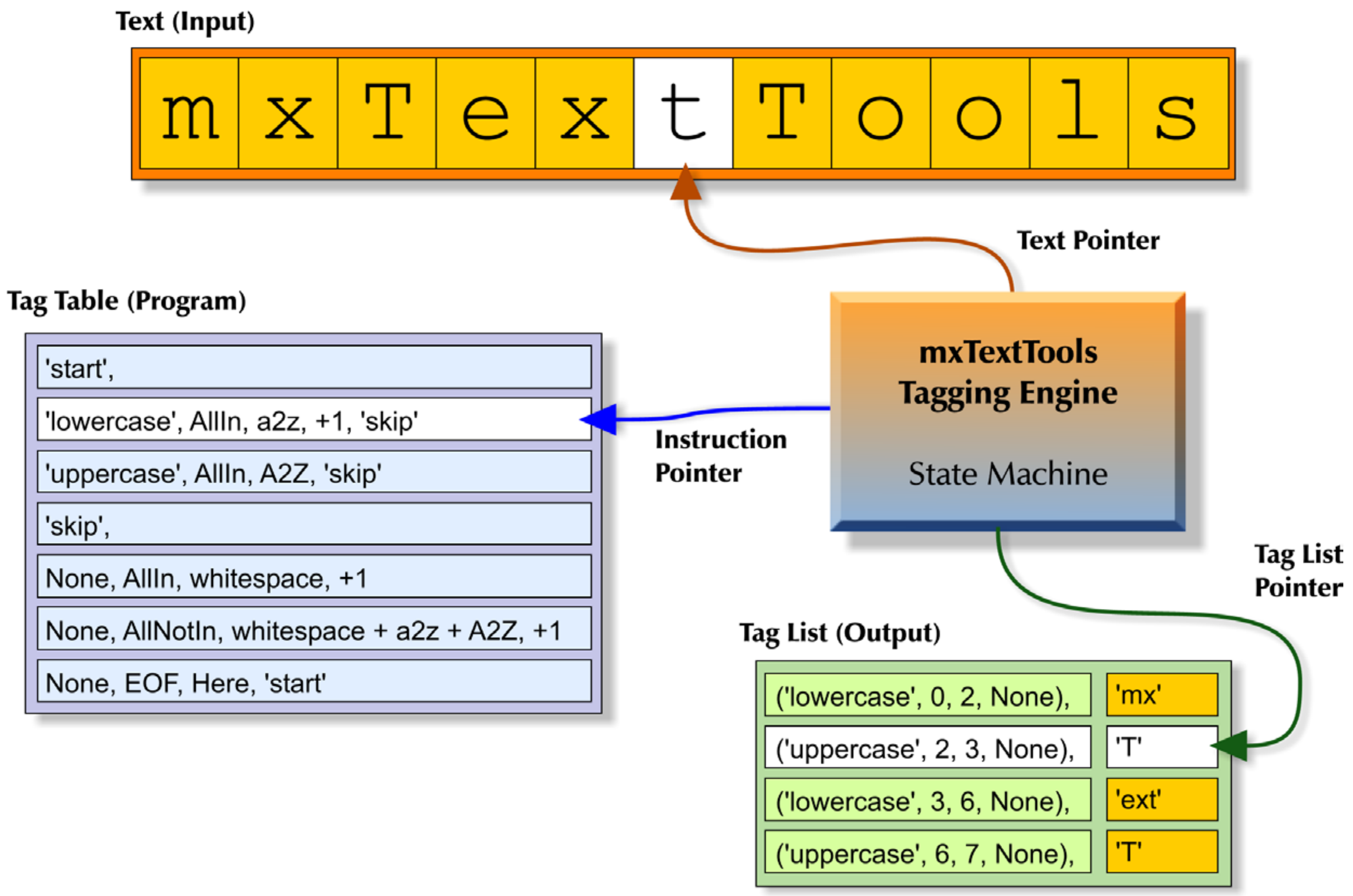
mxTextTools: Key Features

- Tagging Engine
- Search objects
- Helpers for string manipulation
- Helpers for Tagging Engine output
- Full Unicode support !


Text (Input)
m x T e x t T o o l s
Text Pointer
Tag Table (Program)
'start',
'lowercase', AllIn, a2z, +1, 'skip'
'uppercase', AllIn, A2Z, 'skip'
'skip',
None, AllIn, whitespace, +1
None, AllNotIn, whitespace + a2z + A2Z, +1
None, EOF, Here, 'start'
Instruction Pointer
mxTextTools Tagging Engine
State Machine
Tag List Pointer
Tag List (Output)
('lowercase', 0, 2, None), 'mx'
('uppercase', 2, 3, None), 'T'
('lowercase', 3, 6, None), 'ext'
('uppercase', 6, 7, None), 'T'

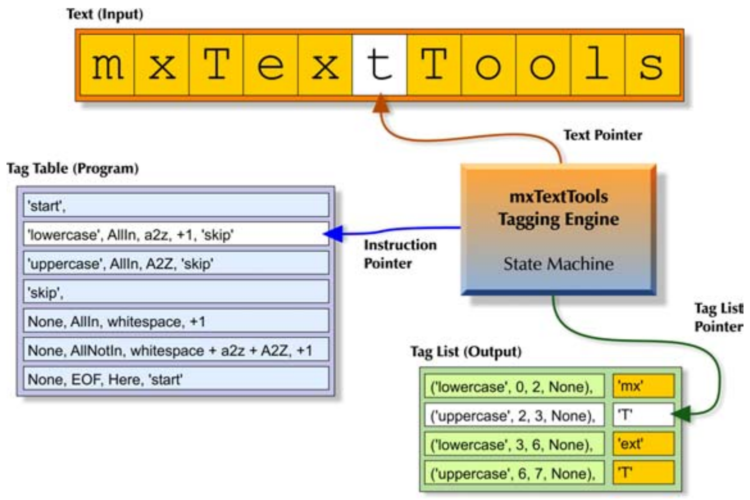
Parsing with the Tagging Engine

Tagging Engine


Text (Input)
m x T e x t T o o l s
Text Pointer
Tag Table (Program)
'start',
'lowercase', AllIn, a2z, +1, 'skip'
'uppercase', AllIn, A2Z, 'skip'
'skip',
None, AllIn, whitespace, +1
None, AllNotIn, whitespace + a2z + A2Z, +1
None, EOF, Here, 'start'
Instruction Pointer
mxTextTools Tagging Engine
State Machine
Tag List Pointer
Tag List (Output)
('lowercase', 0, 2, None), 'mx'
('uppercase', 2, 3, None), 'T'
('lowercase', 3, 6, None), 'ext'
('uppercase', 6, 7, None), 'T'

Tagging Engine: Key Features

- Works on slices (doesn't copy)
- Complete matching command set
- Callbacks to Python
- Arbitrary Tag Objects can be assigned to matching text slices
- Simple API
- JIT compiler
- Full Unicode support



```
'start',
'lowercase', AllIn, a2z, +1, 'skip'
'uppercase', AllIn, A2Z, 'skip'
'skip',
None, AllIn, whitespace, +1
None, AllNotIn, whitespace + a2z + A2Z, +1
None, EOF, Here, 'start'
```

```
('lowercase', 0, 2, None),  'mx'
('uppercase', 2, 3, None),  'T'
('lowercase', 3, 6, None),  'ext'
('uppercase', 6, 7, None),  'T'
```

...fast

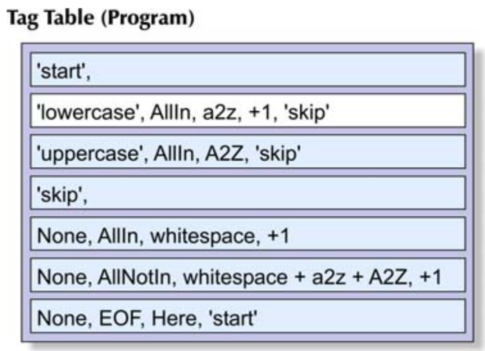
Tag Table Example: Mark text as lower/upper case

```
tag_table = (

    # Tag upper case and lower case text
    'start',
    ('lowercase', AllIn, a2z, +1, 'skip'),
    ('uppercase', AllIn, A2Z, 'skip'),

    # Skip all whitespace & non-letters
    'skip',
    (None, AllNotIn, a2z + A2Z, +1),

    # Check for EOF, otherwise continue
    (None, EOF, Here, 'start'),
)
```


Tag Table (Program)

```
'start',
'lowercase', AllIn, a2z, +1, 'skip'
'uppercase', AllIn, A2Z, 'skip'
'skip',
None, AllIn, whitespace, +1
None, AllNotIn, whitespace + a2z + A2Z, +1
None, EOF, Here, 'start'
```

Compiling Languages with mxTextTools

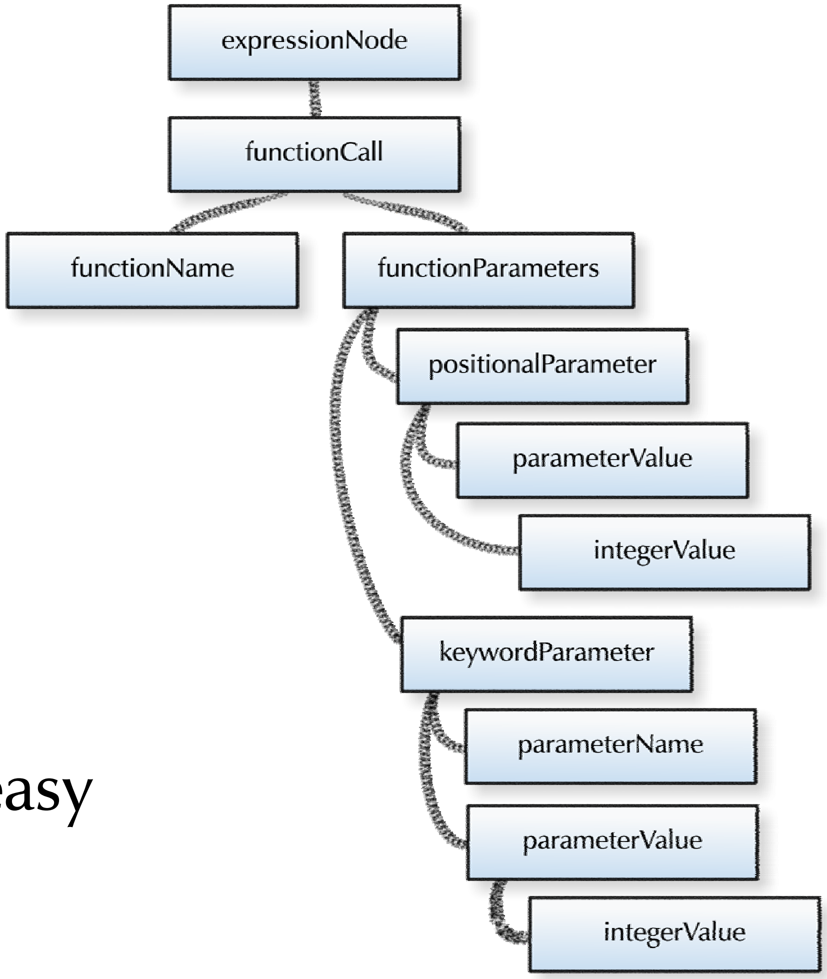
1. Introduction to mxTextTools
2. Parsing with the Tagging Engine
3. Compiling Languages with mxTextTools
4. Discussion

Compiling Languages: General Approach

- Tokenize the input
  - break the input text into logical syntax parts
- Parse the tokens
  - convert/group the tokens to syntax objects
  - group these syntax objects according to logic in a tree
- Manipulate the tree (e.g. to optimize it)
- Traverse the tree and generate a new representation
  - Use the generated representation for further processing

Compiling Languages: Abstract Syntax Trees (ASTs)

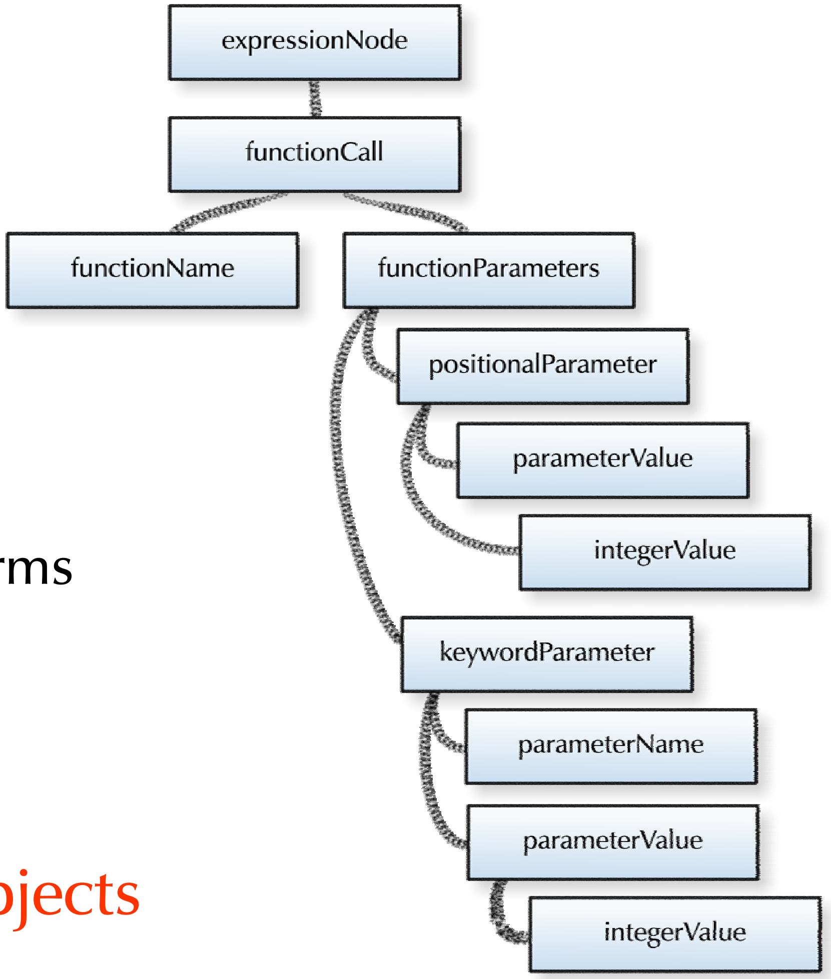
- Provide logical groupings of tokens as objects
- Use the Divide&Conquer approach
- Tree structure makes traversal easy



Compiling Languages: Abstract Syntax Trees (ASTs)

- Node objects contain the "knowledge" about the used syntax
  - have access to context
  - know how to parse tokens and extract their data/meaning
  - can convert the tokens to other forms (e.g. compile them)

- Idea: Use AST classes as Tag Objects



Compiling Languages: From Tag Lists to ASTs

- Recipe:

  1. instantiate the AST class for the first Tag List item (passing it a context object)

  2. pass the remaining Tag List to its .parse() method

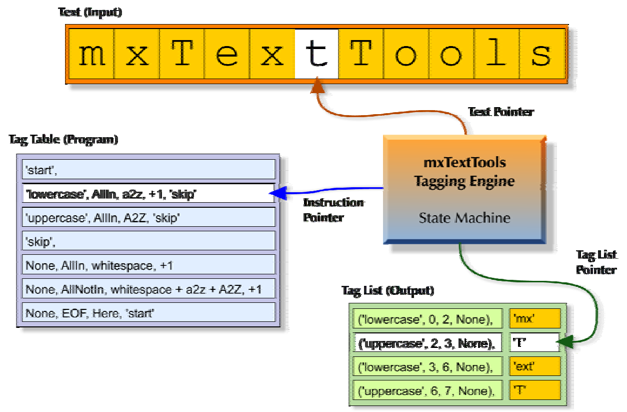
  3. if there are any Tag List items left, go to step 1.

Compiling Languages: From Tag Lists to ASTs

- Recipe:
  1. instantiate the AST class for the first Tag List item (passing it a context object)
  2. pass the remaining Tag List to its .parse() method
  3. if there are any Tag List items left, go to step 1.
- All the parsing logic is put into the hands of the AST objects
  – they can use context information
  – and generate additional information while parsing, e.g. type information

Parsing Languages with mxTextTools: Summary

- Define AST classes to represent the syntax elements
- Define the syntax representation using Tag Tables

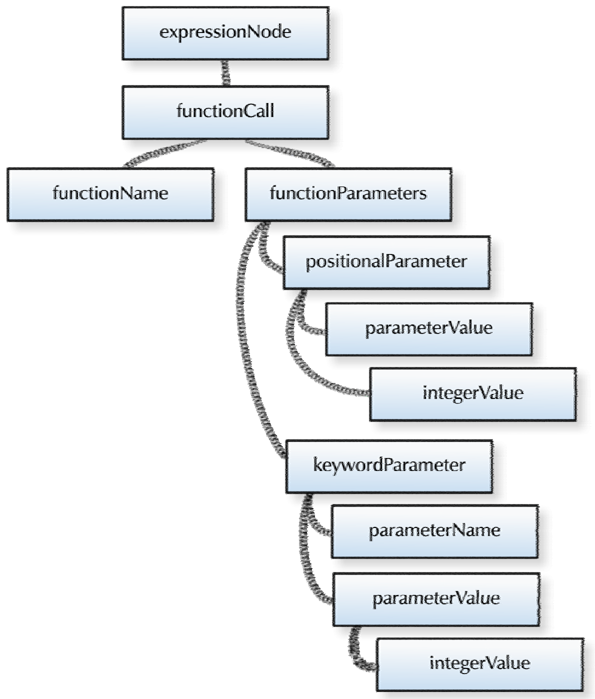

Text (Input)
m x T e x t T o o l s
Text Pointer
Tag Table (Program)
'start',
'lowercase', AllIn, a2z, +1, 'skip'
'uppercase', AllIn, A2Z, 'skip'
'skip',
None, AllIn, whitespace, +1
None, AllNotIn, whitespace + a2z + A2Z, +1
None, EOF, Here, 'start'
Instruction Pointer
mxTextTools Tagging Engine
State Machine
Tag List Pointer
Tag List (Output)
('lowercase', 0, 2, None), 'mx'
('uppercase', 2, 3, None), 'T'
('lowercase', 3, 6, None), 'ext'
('uppercase', 6, 7, None), 'T'

- Use the AST classes as Tag Objects

- Run the Tagging Engine on the input, creating a Tag List

Parsing Languages with mxTextTools: Summary

- Create an AST from the Tag List
- Traverse the AST
- Collect the compiled output



Discussion

Parsing Languages with mxTextTools: Questions ?

And finally...

Thank you for your time.

Contact

eGenix.com Software, Skills and Services GmbH
Marc-André Lemburg
Pastor-Löh-Str. 48
D-40764 Langenfeld
Germany

eMail:	mal@egenix.com
Phone:	+49 211 9304112
Fax:	+49 211 3005250
Web:	http://www.egenix.com/

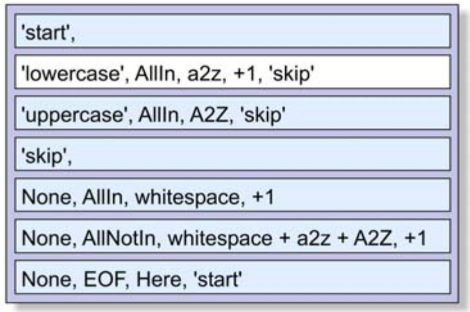
mxTextTools: Availability

- Python package with a highly portable C extension
- eGenix Open-Source Product
- Part of the eGenix mx Base Distribution
- Compiles on:
  - Windows
  - Linux
  - Mac OS X
  - FreeBSD
  - Solaris
  - many other Unix variants

Tagging Engine: Definitions

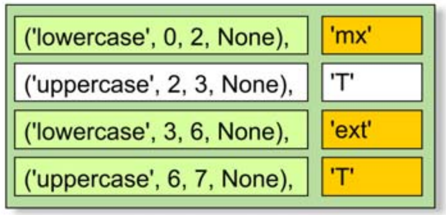
- Tag Table
  - List of tuples defining the matching program
  - Can be nested (e.g. via a Table command)
- Tag Object
  - Object associated with a matching text slice
  - Can be any Python object
- Tag List
  - List of tuples generated by the Tagging Engine
  - Defines the tags

Tag Table (Program)



```
'start',
'lowercase', AllIn, a2z, +1, 'skip'
'uppercase', AllIn, A2Z, 'skip'
'skip',
None, AllIn, whitespace, +1
None, AllNotIn, whitespace + a2z + A2Z, +1
None, EOF, Here, 'start'
```

Tag List (Output)



```
('lowercase', 0, 2, None),   'mx'
('uppercase', 2, 3, None),   'T'
('lowercase', 3, 6, None),   'ext'
('uppercase', 6, 7, None),   'T'
```

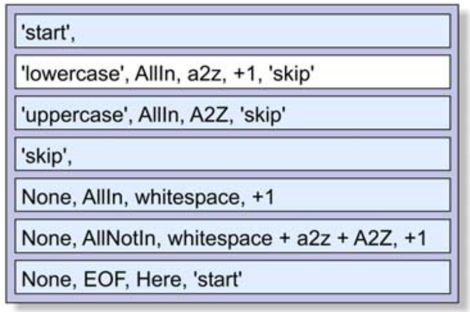
Tagging Engine Input: Tag Table

- Simple and standard item format:

  (tag_object, command, cmd_arg,
  jump_no_match, jump_match)

  with

Tag Table (Program)



```
'start',
'lowercase', AllIn, a2z, +1, 'skip'
'uppercase', AllIn, A2Z, 'skip'
'skip',
None, AllIn, whitespace, +1
None, AllNotIn, whitespace + a2z + A2Z, +1
None, EOF, Here, 'start'
```

- *tag_object* .... object to be associated with the matched slice
- *command* .... command integer
- *cmd_arg* .... command argument
- *jump_no_match* .... jump if not matched (default: leave the table)
- *jump_match* .... jump if matched (default: +1)

Idea: "tag_object is tagged to matching text slice"

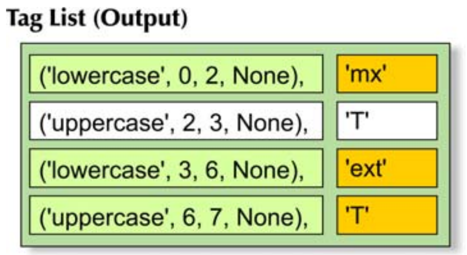
Tagging Engine Output: Tag List

- Simple and standard item format:

  (tag_object, slice_left, slice_right, sub_tag_list)

  with

- *tag_object* …………………. object associated with the slice
- *slice_left, slice_right* …….. slice indexes (text[left:right])
- *sub_tag_list* ………………. None or another Tag List


Tag List (Output)

```
('lowercase', 0, 2, None),    'mx'
('uppercase', 2, 3, None),    'T'
('lowercase', 3, 6, None),    'ext'
('uppercase', 6, 7, None),    'T'
```

Idea: "tag_object is tagged to text[slice_left:slice-right]"

Tag List Example

```
text = "mxTextTools is an extension package for Python..."
```

```
tag_list =                                mx.TextTools.print_tags(text, tag_list)
    [('lowercase', 0, 2, None),               'lowercase' :  'mx' (0, 2)
     ('uppercase', 2, 3, None),               'uppercase' :  'T' (2, 3)
     ('lowercase', 3, 6, None),               'lowercase' :  'ext' (3, 6)
     ('uppercase', 6, 7, None),               'uppercase' :  'T' (6, 7)
     ('lowercase', 7, 11, None),              'lowercase' :  'ools' (7, 11)
     ('lowercase', 12, 14, None),             'lowercase' :  'is' (12, 14)
     ('lowercase', 15, 17, None),             'lowercase' :  'an' (15, 17)
     ('lowercase', 18, 27, None),             'lowercase' :  'extension' (18, 27)
     ('lowercase', 28, 35, None),             'lowercase' :  'package' (28, 35)
     ('lowercase', 36, 39, None),             'lowercase' :  'for' (36, 39)
     ('uppercase', 40, 41, None),             'uppercase' :  'P' (40, 41)
     ('lowercase', 41, 46, None)]             'lowercase' :  'ython' (41, 46)
```

Tag List Example

```
text = "mxTextTools is an extension package for Python..."
```

```
tag_list =                              mx.TextTools.print_tags(text, tag_list)
    [('lowercase', 0, 2, None),             'lowercase' :  'mx' (0, 2)
     ('uppercase', 2, 3, None),             'uppercase' :  'T' (2, 3)
     ('lowercase', 3, 6, None),             'lowercase' :  'ext' (3, 6)
     ('uppercase', 6, 7, None),             'uppercase' :  'T' (6, 7)
     ('lowercase', 7, 11, None),            'lowercase' :  'ools' (7, 11)
     ('lowercase', 12, 14, None),           'lowercase' :  'is' (12, 14)
     ('lowercase', 15, 17, None),           'lowercase' :  'an' (15, 17)
     ('lowercase', 18, 27, None),           'lowercase' :  'extension' (18, 27)
     ('lowercase', 28, 35, None),           'lowercase' :  'package' (28, 35)
     ('lowercase', 36, 39, None),           'lowercase' :  'for' (36, 39)
     ('uppercase', 40, 41, None),           'uppercase' :  'P' (40, 41)
     ('lowercase', 41, 46, None)]           'lowercase' :  'ython' (41, 46)
```

Tagging Commands

- Character and word matching
- Character set matching

- Jumps in text and Tag Table
- Recursive matching (using nested Tag Tables)
- Callbacks to Python (e.g. to do more complicated matching)
- String jump targets

Idea: “Highly optimized, with everything you need for parsing”