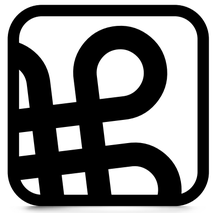


# Macoun

# Wie man eine neue Programmiersprache entwickelt

Karsten Kusche - 2014

@_karsten_

Briksoftware.com

# Smalltalk

# Programmiersprachen

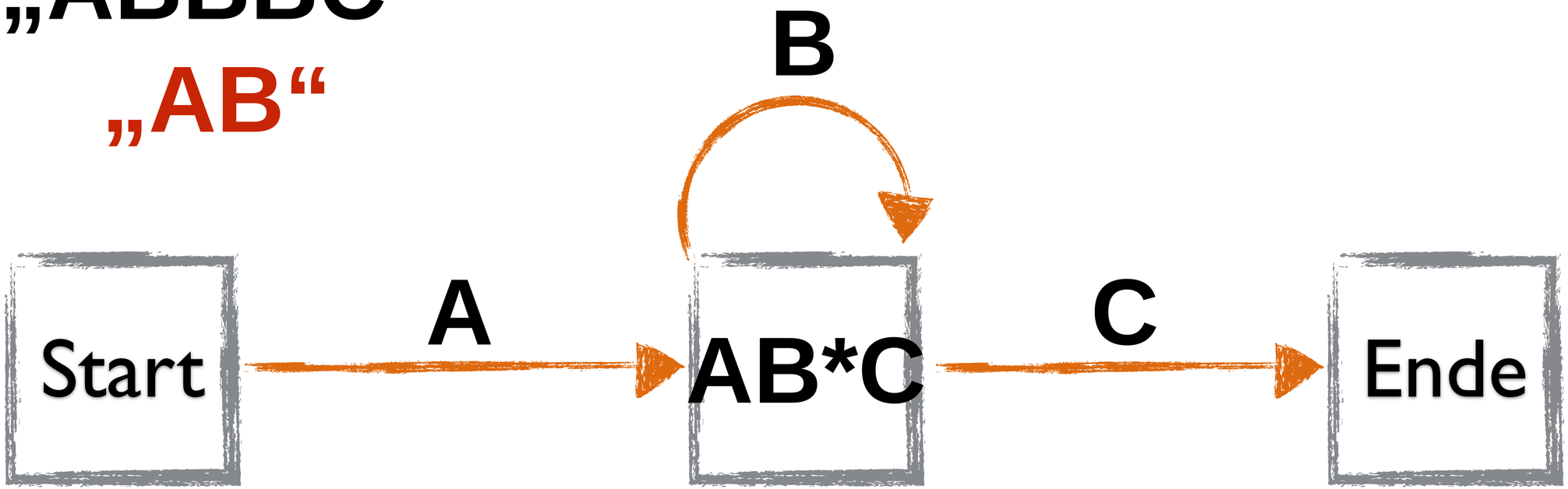
# Automaten

„AB“

„ABC“

„ABBBC“

„AB“



# WAT

[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}

# Palindromproblem

((LageBrand)*C)

# Grammatiken

## BNF

```
Gleichung ::= Operator Operation Operator
Operator ::= Zahl | '(' Gleichung ')'
Operation ::= '+' | '-'
Zahl ::= '1' | '2' | … | '9' | '0'
```

Parser

```
Gleichung ::= Operator Operation Operator
Operator ::= Zahl | '(' Gleichung ')'
```

Scanner

```
Operation ::= '+' | '-'
Zahl ::= '1' | '2' | … | '9' | '0'
```

Tokens

# YACC

* Compiler Compiler

# PEGKit

Parsing Expression Grammar

# PEGKit

github.com/itod/pegkit

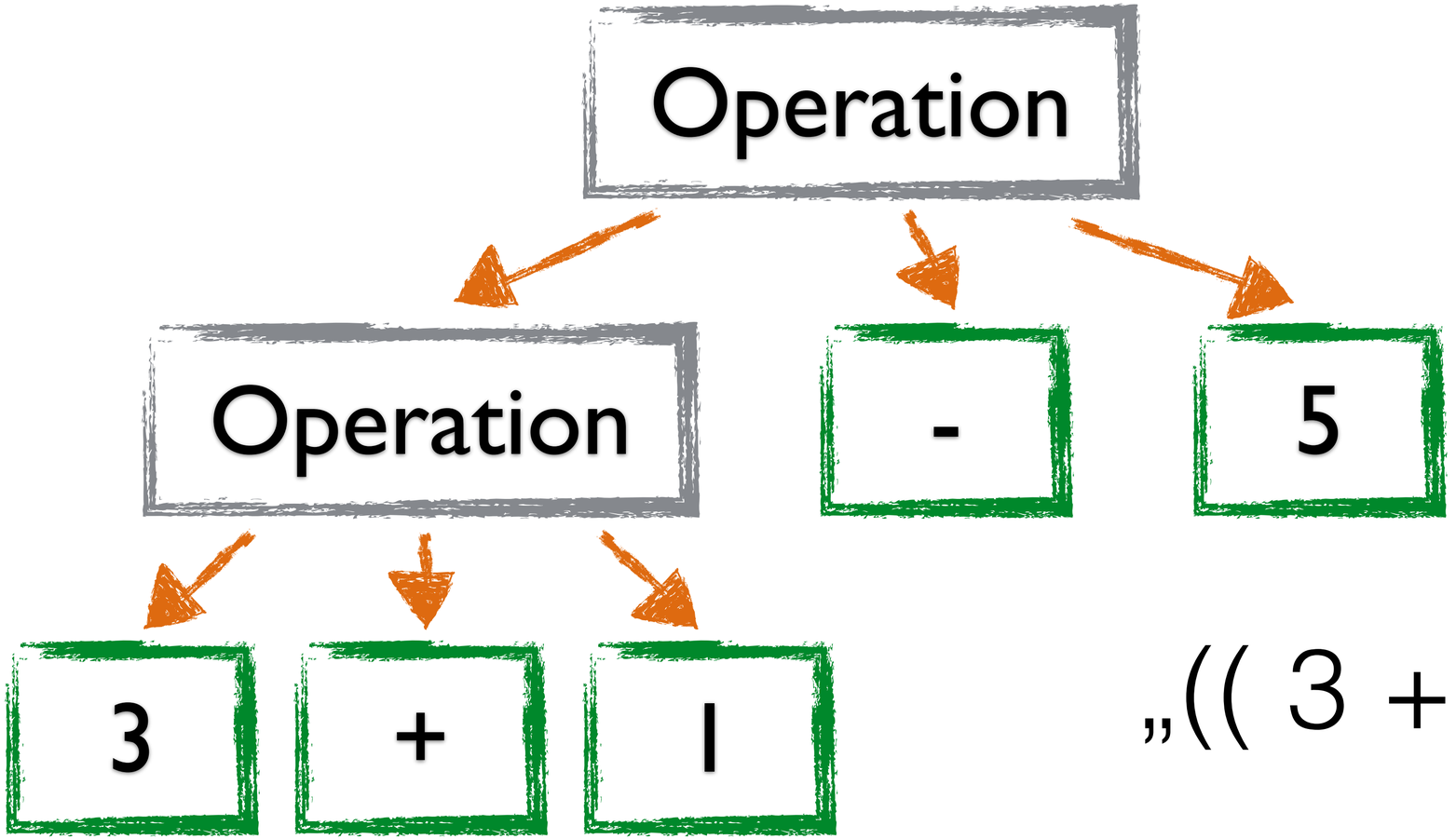
```
start = gleichung | operator;
gleichung = operator operation operator;
operator = Number | '(' gleichung ')';
operation = '+' | '-';
```

„(( 3 + 1 ) - 5 )“

„(( 3 + 1 ) - 5 )“

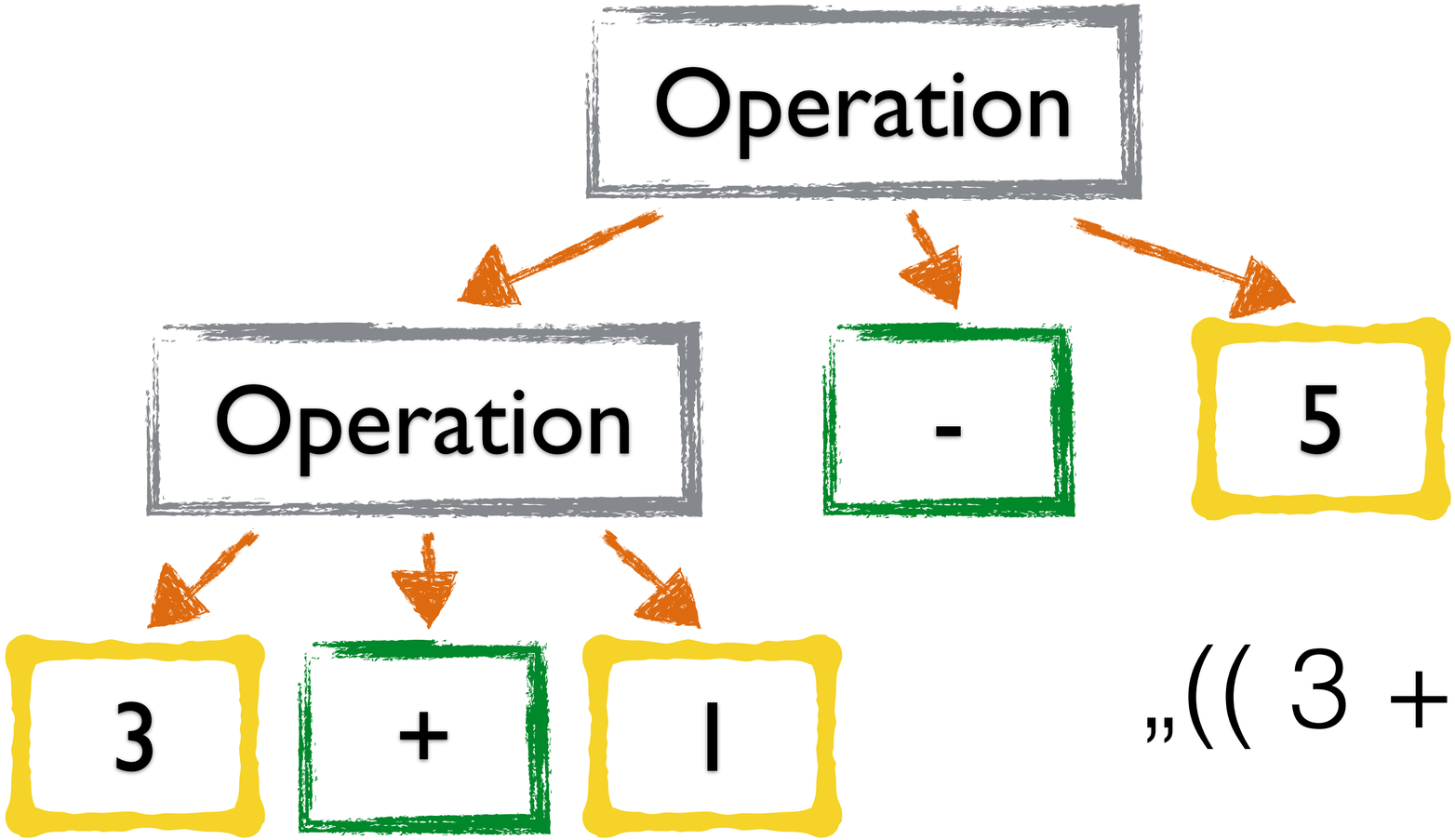
```
operator = Number | '('! gleichung ')'!;
```

```
gleichung = operator operation operator {
   PUSH([Operation rechts: POP()
                  operator: POP()
                     links: POP()]);
}
```




„(( 3 + 1 ) - 5 )“

```
operator = Number {
   NSInteger zahl = POP_INT();
   PUSH([NSNumber numberWithInt:zahl]);
}
```




„(( 3 + 1 ) - 5 )“

# Baloo

**anweisung** *argument, argument*

```
zeile = anweisung argumente | anweisung;
argumente = argument ','! argumente |
argument;
```

```
zeile = anweisung argumente | anweisung;
```

VS

```
zeile = anweisung | anweisung argumente;
```

# Argumente

*"Hallo Welt", 2, YES, NO, 5*

```
argument = literal;
literal = QuotedString | Number | bool;
bool = 'YES' | 'NO';
```

# Variablen

```
argument = literal | variable;
literal = QuotedString | Number | bool;
bool = 'YES' | 'NO';
variable = Word;
```

```
add summe, 4, 5
nslog "Summe: %@", summe
```

# Programm-Strukturen

# C

```c
#include <stdio.h>
typedef struct point { int x, int y} point;
int main()
{
  int a;
  if (1) {
     printf("Enter an integer\n");
     scanf("%d", &a);
  }
  printf("Integer that you have entered is %d\n", a);

  return 0;
}
```

# Objective-C

```
#import <Foundation/Foundation.h>

@interface Person : NSObject {
    NSDate* birthday;
}
@property (retain) NSString* name;

+ (instancetype)karsten;

@end
```

# Objective-C

```
- (void)sayHi
{
    @try {
        [self say: @"Hi!"];
    }
    @catch (NSException *exception) {
        NSLog(@"Error:%@", exception);
    }
    @finally {
        NSLog(@"Done!");
    }
}
```

# Smalltalk

```
Smalltalk.Macoun defineClass: #Example
   superclass: #{Object}
   indexedType: #none
   private: false
   instanceVariableNames: 'name description '
   classInstanceVariableNames: ''
   imports: ''
   category: ''
```

# Smalltalk

```
say: aString toAll: people
   people do:[:each |
      [self say: aString to: each]
         on: DoesntListen , IsIgnorant
         do:[:exception |
            self yell: aString to: each.
         ]
   ]
```

```
add summe, 4, 5
nslog "Summe: %@", summe
```

```
add summe, 4, 5
greaterThan isBigger, summe, 10
ifTrue isBigger
{
   nslog "Große Summe: %@", summe
}
```

```
zeilen = (zeile | block) cr zeilen | zeile
| block;
blockRule = '{'! cr zeilen cr '}'!
| '{'! cr '}'!;
cr = ('\n'! | '\r'!)+;
```

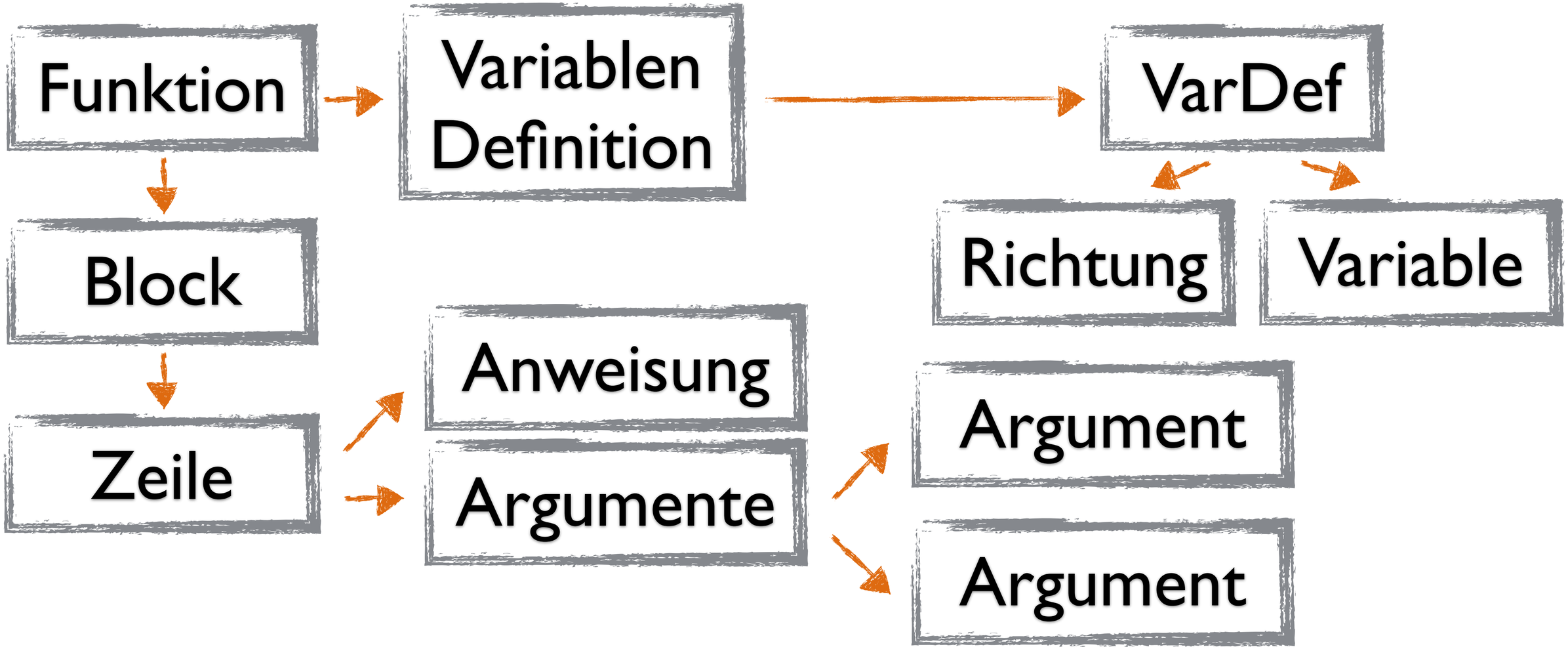
```
funktion = Word variablenDefinition cr blockRule
   | Word cr blockRule;
variablenDefinition = varDef cr variablenDefinition
   | varDef;
varDef = richtung ':'! variable;
richtung = 'ein' | 'aus' | 'tmp';
```

# AST

```
argument = literal {
   PUSH([ArgumentNode withLiteral: POP()]);
};
literal = QuotedString {
   PUSH([LiteralStringNode with:POP()]);
}
| Number {
   PUSH([LiteralNumberNode with:POP()]);
}
...
```

```
argumente = argument ','! argumente {
  ArgumentsNode* args = POP();
  ArgumentNode* arg = POP();
  [args addFirstArgument:arg];
  PUSH(args);
}
| argument {
  PUSH([ArgumentsNode withArgument: POP()]);
};
```

```
@m {
#import "AllNodesClasses.h"
}
@before {
    PKTokenizer *t = self.tokenizer;
    [t setTokenizerState:t.symbolState from:'\\n' to:'\\n'];
    [t setTokenizerState:t.symbolState from:'\\r' to:'\\r'];
}
```


Funktion
Variablen Definition
VarDef
Block
Richtung
Variable
Anweisung
Zeile
Argumente
Argument
Argument

# AST Verarbeitung

# Tests

# Autoformatierung - Einfärbung

# Automatische Hilfe

# Code Validierung

# Code Transformationen

# Code Suche

# Code Ausführung

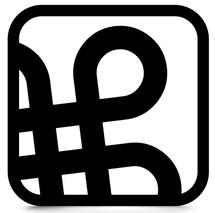
# Code Speicherung

# Code Bearbeitung

# Parser-Leistung

# Fragen?

# Vielen Dank



# Macoun