# The Pyttern Program Query Language

Julien Liénard

Kim Mens

Siegfried Nijssen

SCLIT 25

# Context

Student make mistakes

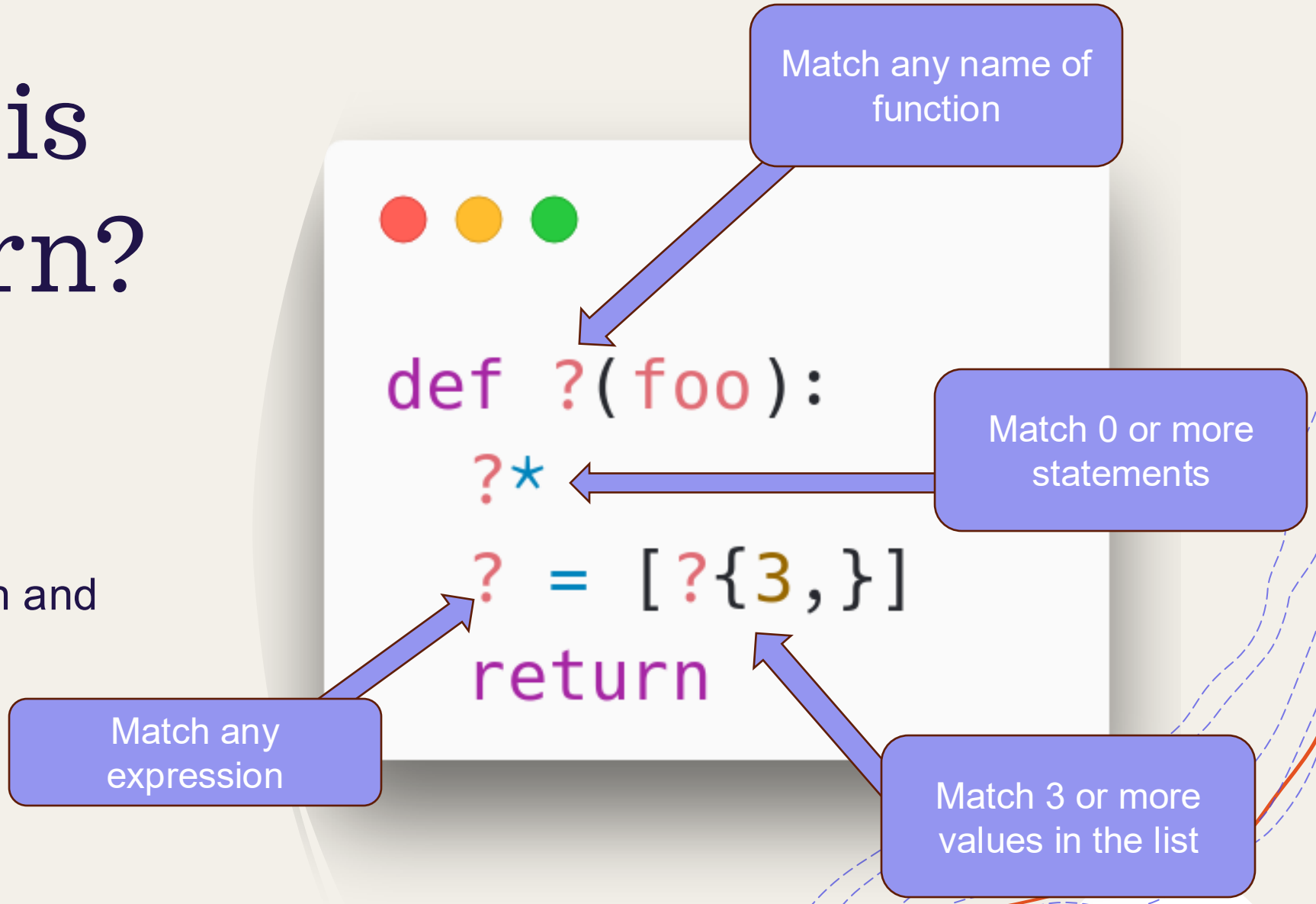
Student have misconception and bad smells in their code

How to detect them efficiently?

Lots of PQL but can be hard to learn

# What is Pyttern?

- Language to describe patterns
- Build over Python
- Build to be easy to learn and use
- Inspired by regex



# Some examples

Match any number of arguments

```
def facteurs(?*):
    ?:*
        for ? in range(?*):
            if ? == 0:
                ?*
```

Match the body at any level of identation

Match only While or For expressions

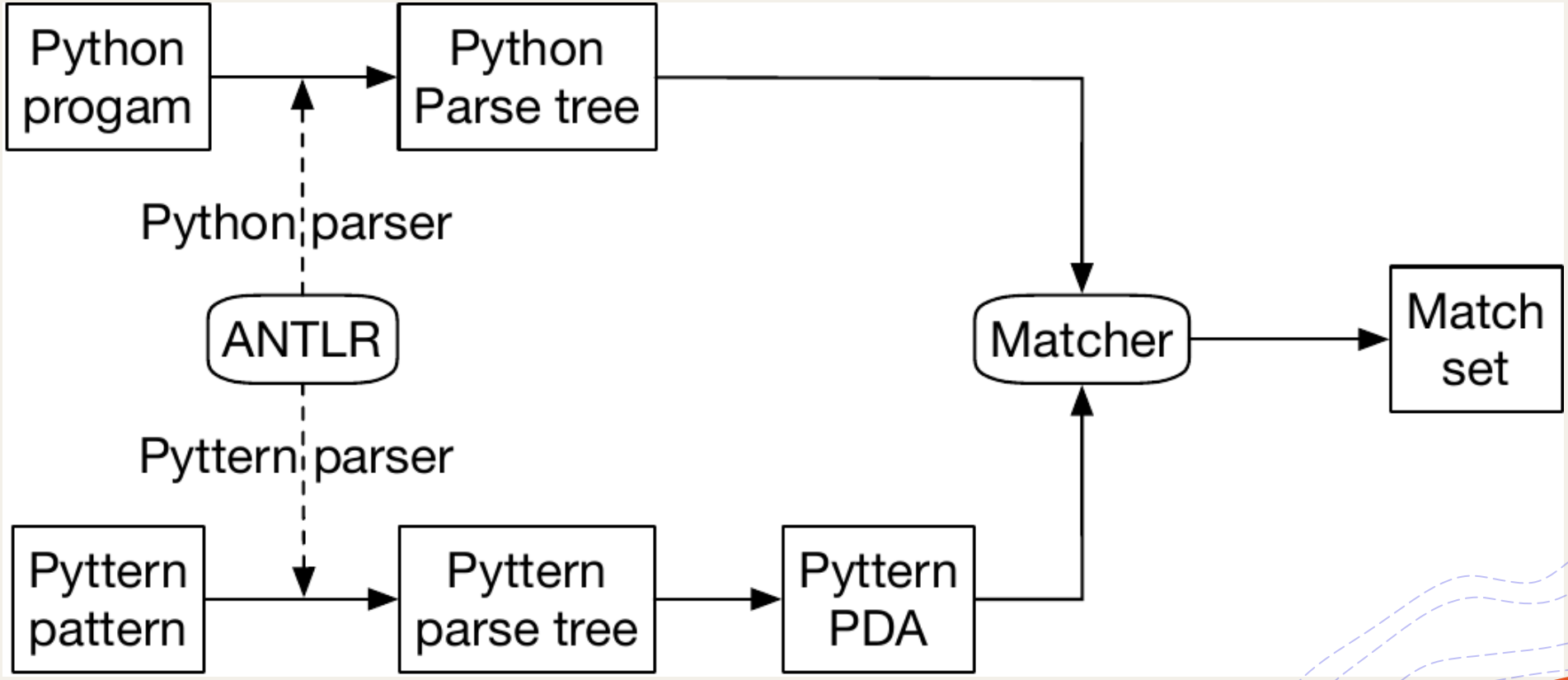
```
def ?(?*):
    ?*
    ?[While, For]:*
        ?*
```

```
def approx_pi(n):
    ?pi = 0
    ?:*
        ?pi = ?pi + ?
    return ?pi
```

Bind the match to the variable "pi"

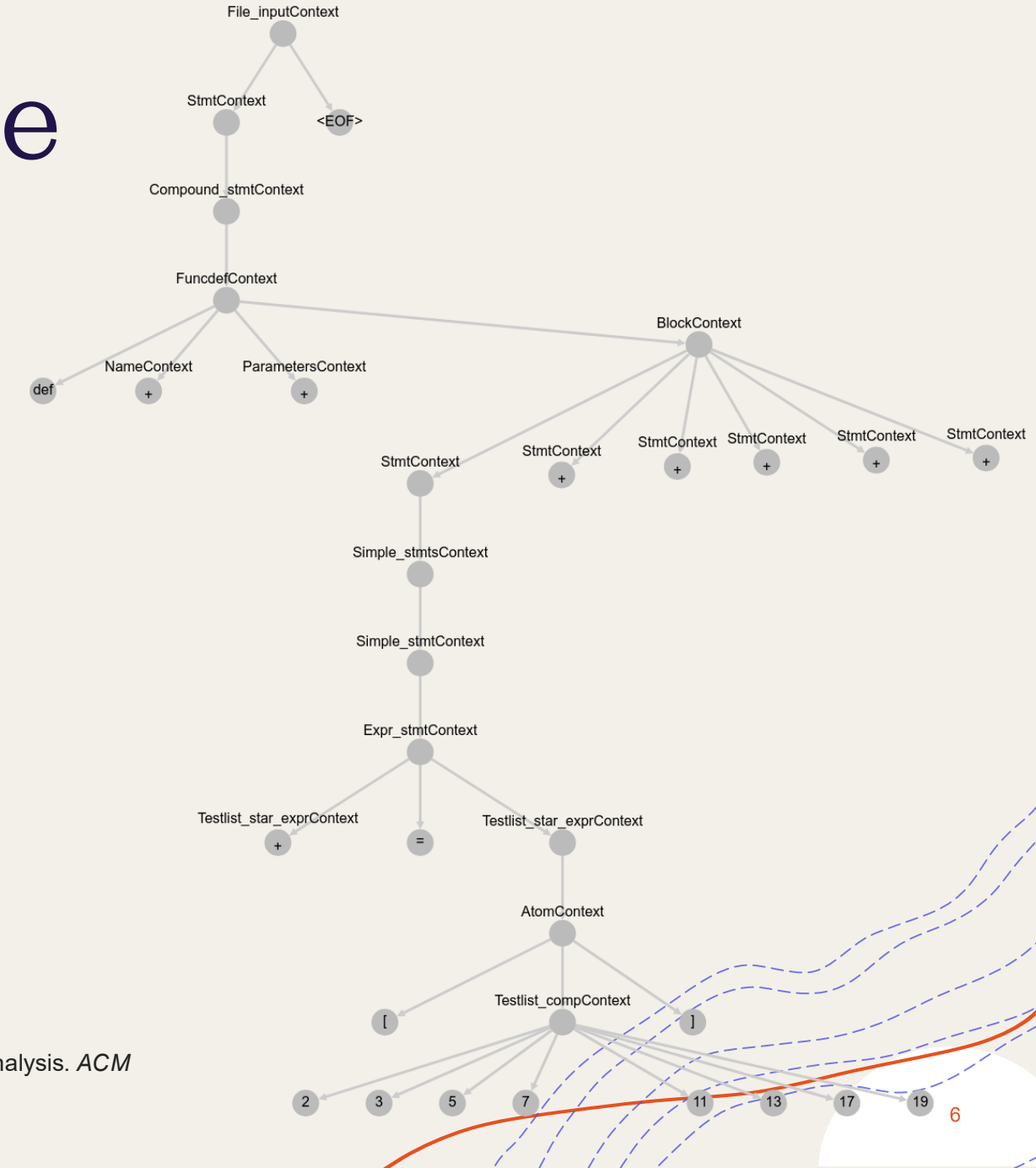
# Architecture

+ Based on SCRUPLE
+ Designed for adaptability



# ANTLR Parse Tree

- Adaptive LL(*) parser [1]
- Easy to use
- Base Python grammar already defined
- Easy to find grammar for other languages

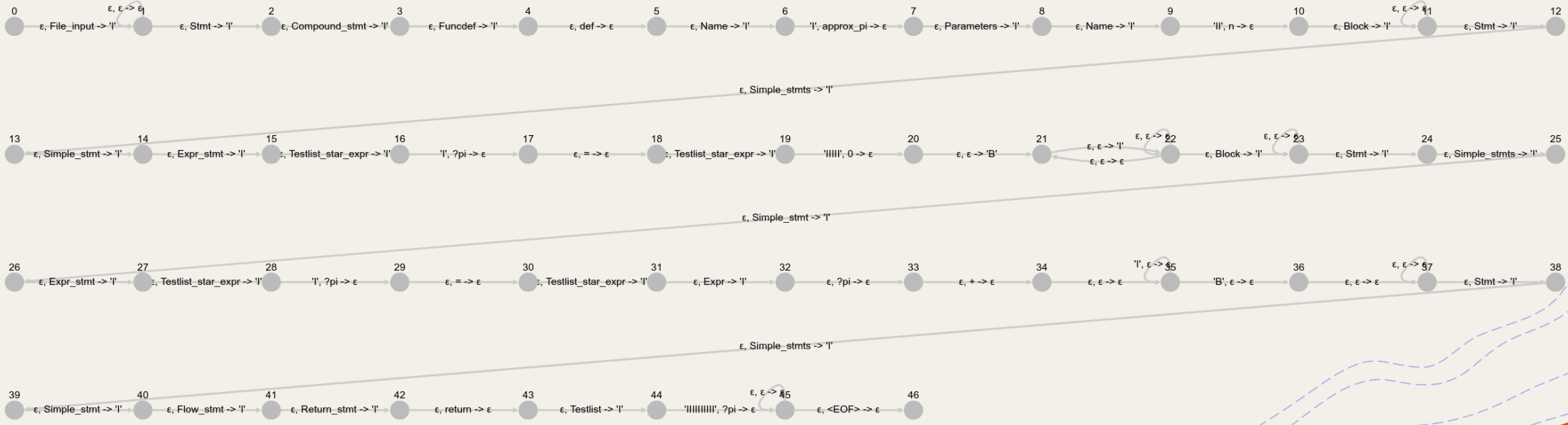




1. Parr, T., Harwell, S., & Fisher, K. (2014). Adaptive LL (*) parsing: the power of dynamic analysis. *ACM SIGPLAN Notices*, *49*(10), 579-598.

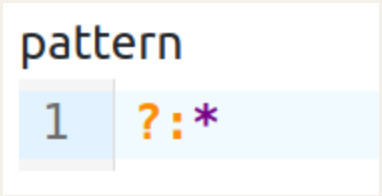
# Pyttern Custom PushDown Automaton

- Non-deterministic PDA
- Determine how we traverse the parse tree
- Constraint on tree traversal to ensure termination

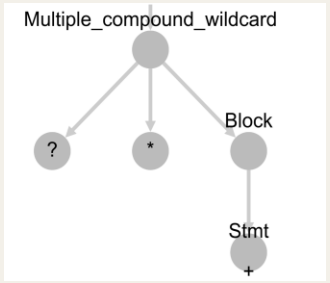


# Pyttern to PDA

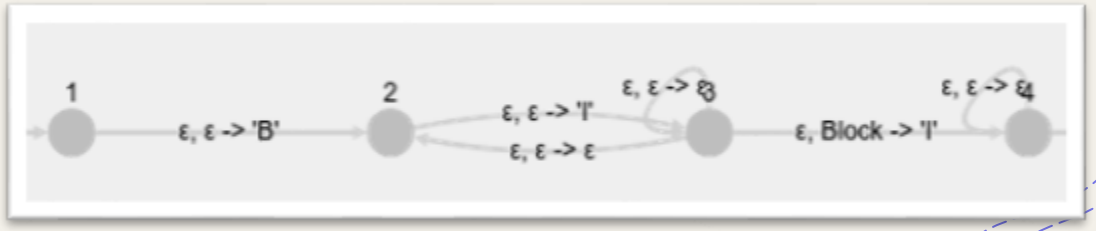
- Pyttern parsed with ANTLR
- Custom visitor to generate the PDA
- Each wildcards generate a specific PDA structure




Pyttern




Parse Tree




PDA

# Pyttern Custom PDA

- The custom PushDown Automaton is a 6-tuple $(Q, \Sigma, T, \delta, q_0, q_f)$:
  - Q is a finite set of states generated by Pyttern
  - $\Sigma$ is the alphabet of labels of the nodes in the trees
  - T is an additional alphabet representing pyttern labelled wildcards ($\Sigma \cap T = \emptyset$)
  - $\delta$ is the transition relations
  - $q_0 \in Q$ is the start state
  - $q_f \in Q$ is the end state

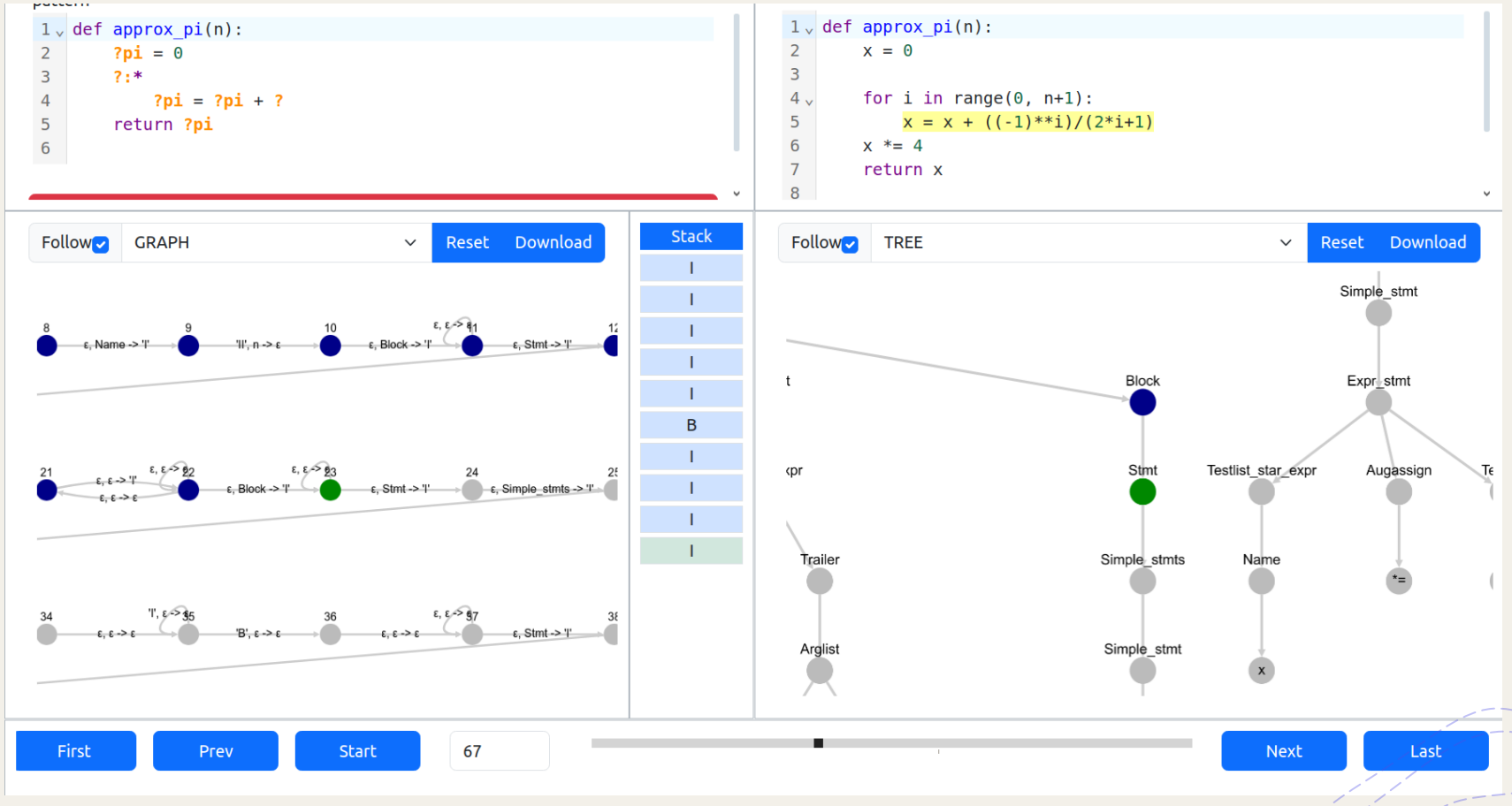
# Pyttern Custom PDA

- Each transition is a 6-tuple $(q, a, A, t, q', \alpha) \in \delta$:
  - $q \in Q$ is the current state
  - $a \in \Sigma \cup T \cup \{\epsilon\}$, indicating a label or labelled wildcard condition on the current node in the input tree
  - $A \in \{\epsilon, B, I\}$ indicates a condition on a symbol on the top of the stack
  - $t \in$ {left-child, right-sibling, parent} is the navigation direction
  - $q' \in Q$ is the next state
  - $\alpha \in \{\epsilon, B, I\}$ is the stack symbol to push to the stack

# Pyttern Custom PDA

- The PDA uses transitions to move from one configuration to another.
- A configuration is defined by a 4-tuple (q, v, s, m):
  - $q \in Q$ is a state in the PDA
  - $v \in V$ is the current node in the Python parse tree, V is the set of nodes of the parse tree
  - $s \in \{B, I\}*$ is a string of symbols currently on the stack
  - $m : T \rightarrow V \cup \{\epsilon\}$ is a mapping from named wildcards to a node in the Python parse tree

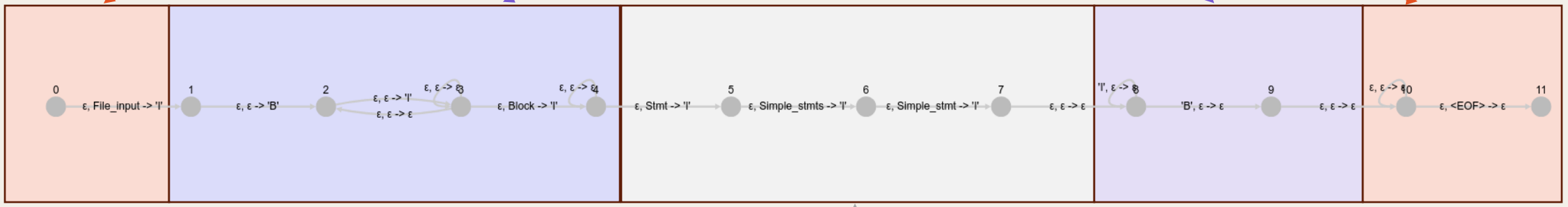
# Matching


def approx_pi(n):
?pi = 0
?:*
?pi = ?pi + ?
return ?pi
def approx_pi(n):
x = 0
for i in range(0, n+1):
x = x + ((-1)**i)/(2*i+1)
x *= 4
return x
Follow
GRAPH
Reset
Download
Stack
I
I
I
I
I
B
I
I
I
I
Follow
TREE
Reset
Download
Simple_stmt
Block
Expr_stmt
Stmt
Testlist_star_expr
Augassign
Simple_stmts
Name
Simple_stmt
First
Prev
Start
67
Next
Last

# Matching

Same for every pattern

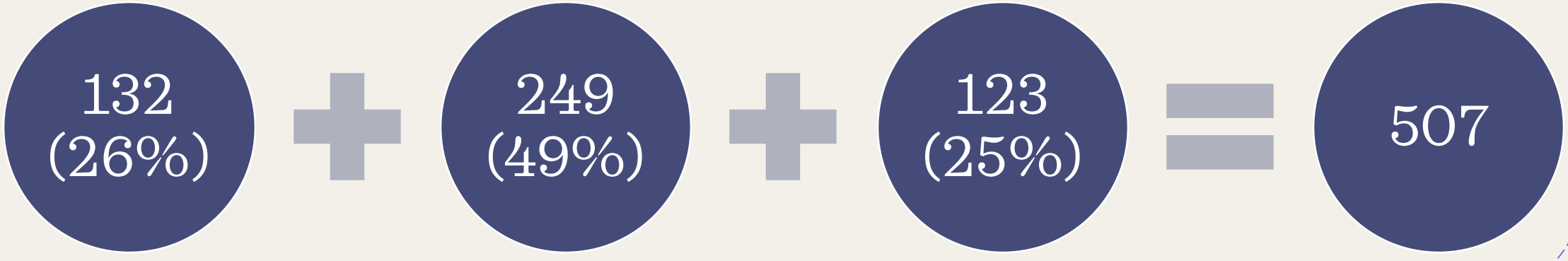
Structure for matching ?:*




Match what is inside ?:*

# Validation

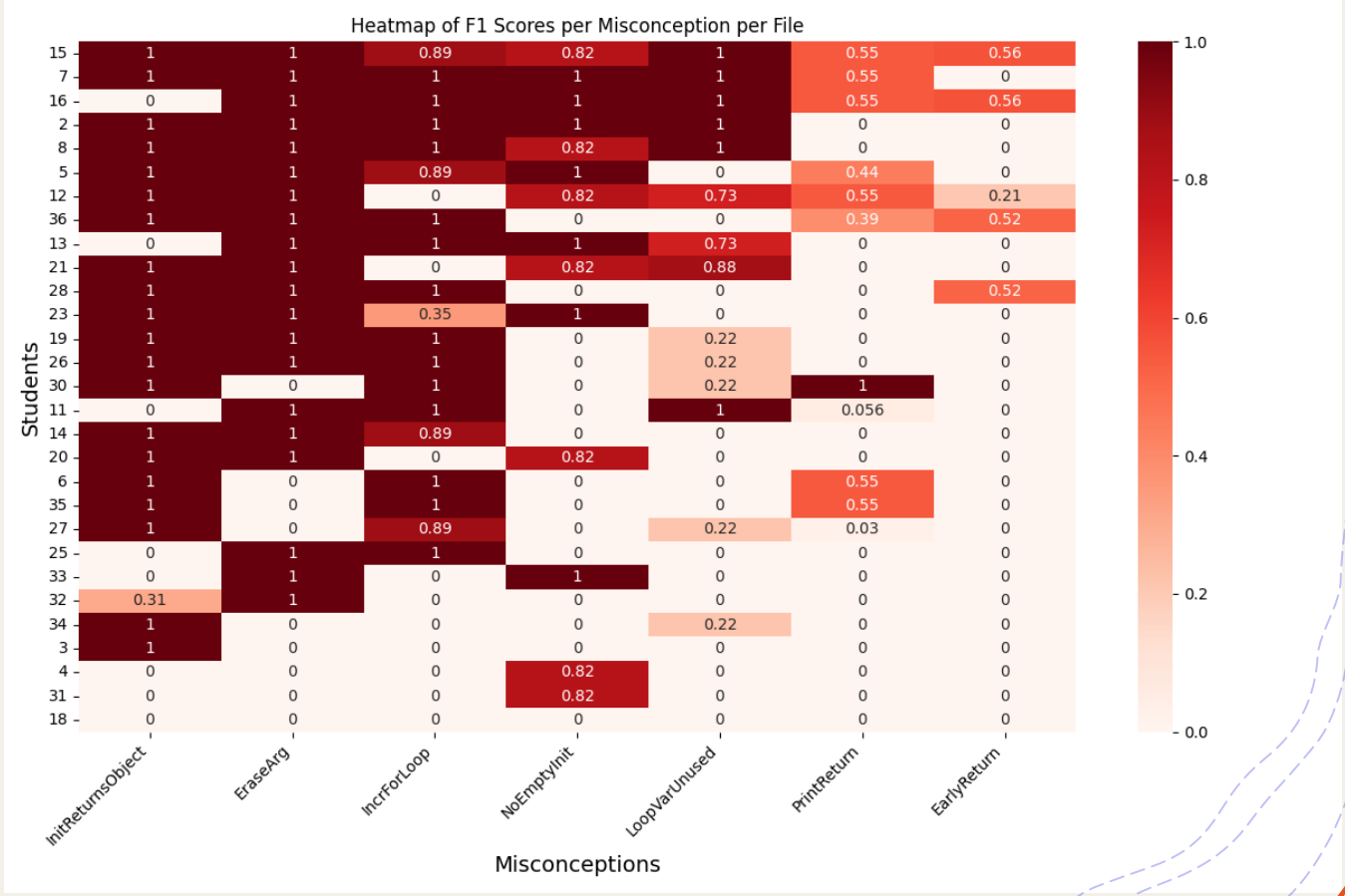
+ Generated files
+ Started from correct code
+ Adding misconception using technique described by Steveny et al. [1]
+ Allows to keep track of the misconceptions in the files




1. Guillaume Steveny, Julien Liénard, Kim Mens, and Siegfried Nijssen. Feedback with bert: When detecting students' misconceptions becomes automatic. In Responsible Knowledge Discovery in Education (RKDE), 2024.

# Validation

- 35 Master students
  - 29 wrote at least 1 pyttern
  - 28 matches at least 1 file
- Good learning curve
- Some problem with matching using multiple patterns



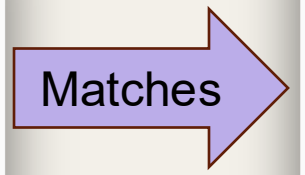
# Conclusion

+ Language with simple wildcards
+ Strong foundation with formalism
+ Designed as framework for other languages
+ Good learning curve

# Next steps

+ Implementation in Java
+ Adding more features
  - "not" wildcard
  - Syntactic macros to match multiple structures

```
def ?(?*):
  ?loop i over 10:
    ?<i>
```




```
def foo():
  i = 0
  while i < 10:
    print(i)
```

```
def foo():
  for i in range(10):
    print(i)
```