Formalizing a Language

Giving structure to raw text

Comprehending Python

- Python is read by humans
  - Parsed similar to a natural language
- `up = requests.get("https://status.python.org")`
  - up equals requests dot get https status dot python dot org
  - up gets the value of calling the get method of requests with the string https colon forward slash forward slash status dot python dot org

Comprehending Python

- Python is read by computers
  - Parsed into a syntax tree
- `up = requests.get("https://status.python.org")`
  - ```
    Module(body=[Assign(targets=[Name(id='up')],
       value=Call(
         func=Attribute(value=Name(id='requests'),attr='get'),
         args=[Constant(value='https://status.python.org')]))
     ])
    ```
  - `01100101110100000011001001010000101011010011001 00`
    ```

Compiling Python

Compiling Python

- Lexing
  - Turn a stream of characters into a stream of tokens
  - Tells the computer what the words are
-
```
up = requests.get("https://status.python.org")
^^ ^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
|  | |       ||  ||                          ||
|  OP|      OP| OP|                         OP|
NAME NAME     NAMESTRING                NEWLINE
```

Compiling Python

- Parsing
  - Turn a stream of tokens into a syntax tree
  - Grammar defines the set of valid programs
- `up = requests.get("https://status.python.org")`
  - ```
    Assign(
      targets=[Name(id='up')],
      value=Call(
        func=Attribute(value=Name(id='requests'),attr='get'),
        args=[Constant(value='https://status.python.org')]
      )
    )
    ```
    ```

Compiling Python

- Assembling
  - Turn a syntax tree into bytecode
  - Can be executed in the cpython virtual machine
  - Can be saved as a pyc file
- `up = requests.get("https://status.python.org")`
  - `01100101110100000011001001010000101011010011001 00`

Formal Grammar

Formal Grammar

- Set of rules that define what combinations of words are valid for a language
  - `up = requests.get("https://status.python.org")`
    - Lexically valid
    - Syntactically valid
  - `up := requests-get{"https://"++"status.python.org"}`
    - Lexically valid
    - Not syntactically valid

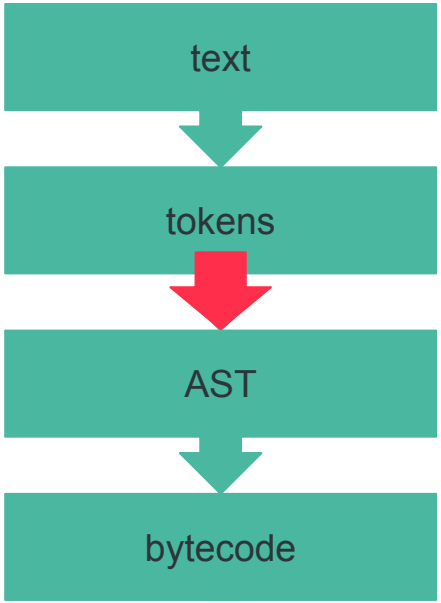
Formal Grammar

- Python <= 3.9
  - Deterministic context-free grammar
  - EBNF notation
  - LL(1) parser

Formal Grammar

- Python >= 3.9
  - Parsing expression grammar
  - Python notation
  - Packrat parser

Formal Grammar



Defining a Grammar

EBNF molecule

```
power: atom_expr ['**' factor]
atom_expr: [AWAIT] atom trailer*
atom: ('(' [yield_expr|testlist_comp] ')' |
       '[' [testlist_comp] ']' | '{' [dictorsetmaker] '}' |
       NAME | NUMBER | STRING+ | '...' | 'None' |
       'True' | 'False')
trailer: '(' [arglist] ')' | '[' subscriptlist ']' | '.' NAME
```

PEG molecule

```
power: await_primary '**' factor | await_primary
await_primary: AWAIT primary | primary
primary: primary '.' NAME | primary genexp
    | primary '(' [arguments] ')' | primary '[' slices ']'
    | atom
atom: NAME | 'True' | 'False' | 'None' | &STRING strings
    | NUMBER | &'(' (tuple | group | genexp)
    | &'[' (list | listcomp)
    | &'{' (dict | set | dictcomp | setcomp) | '...'
```

EBNF arguments

```
arglist: argument (',' argument)*  [',']
argument: ( test [comp_for] |
            test ':=' test |
            test '=' test |
            '**' test |
            '*' test )
```

PEG arguments

```
arguments: args [','] &')'
args: ','.(starred_expression | named_expression !'=')+ [',' kwargs]
    | kwargs
kwargs: ','.kwarg_or_starred+ ',' ','.kwarg_or_double_starred+
    | ','.kwarg_or_starred+ | ','.kwarg_or_double_starred+
starred_expression: '*' expression
kwarg_or_starred: NAME '=' expression | starred_expression
kwarg_or_double_starred: NAME '=' expression | '**' expression
```

The Bright Future of PEG

PEG with

```
with_stmt:
    | 'with' '(' ','.with_item+ ','? ')' ':' block
    | 'with' ','.with_item+ ':' [TYPE_COMMENT] block
with_item:
    | expression 'as' star_target &(',' | ')' | ':')
    | expression
```

EBNF with

```
with_stmt: 'with' with_item (',' with_item)*  ':'
    [TYPE_COMMENT] suite
with_item: test ['as' expr]
```

Soft Keywords

- Keywords
  - `def True lambda`
- Reserved words
  - `Ellipsis NotImplemented`
- Soft keywords
  - `match case`

Soft Keywords

- Wait! async await were both soft keywords
- Only by changing the tokenizer

```
function_def_raw: 'def' NAME '(' params=[params] ')'
    ['->' expression] &&':' [func_type_comment] block
  | ASYNC 'def' NAME '(' params=[params] ')'
    ['->' expression] &&':' [func_type_comment] block
match_stmt: "match" subject_expr ':' NEWLINE INDENT
    case_block+ DEDENT
```

PEG Wins

- Grammar alone defines what is correct syntax
  - And so also syntax errors
- Parser generates AST directly
  - Rules recursion structurally matches the syntax tree
- Soft keywords
  - `match` and `case` will likely not transition to hard keywords
  - More will be coming
- More expressive

Thanks!

- EuroPython
- Volunteers
- ActiveState
- PSF
- Python Community

Questions?