

DSL в Python.

Как и зачем?

Цыганов Иван
Positive Technologies

Предметно ориентированный язык — это
язык программирования с ограниченными
выразительными возможностями,
ориентированный на некую конкретную
предметную область

— *Мартин Фаулер*

SQL

```
SELECT * FROM Users WHERE Age>20
```

SQL REGEXP

[A-Z] [A-Za-z0-9]*

SQL REGEXP LaTeX

E = mc²\

SQL
REGEXP
LaTeX
HTML

`PiterPy`

SQL
REGEXP

LaTeX

HTML

PonyORM

`select(p for p in Person if p.age > 20)`

SQL
REGEXP

LaTeX

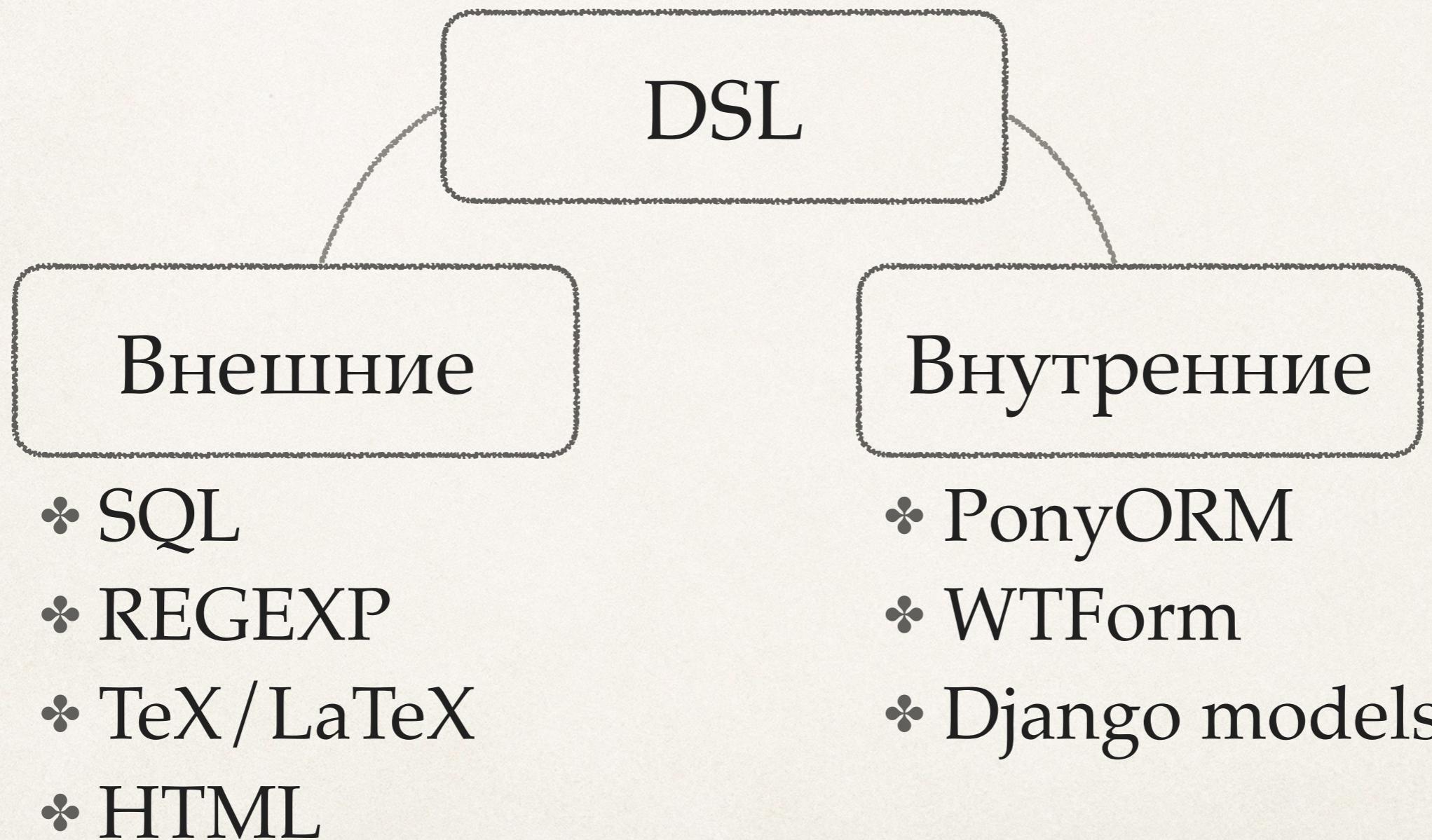
HTML

PonyORM

WTForms

```
class UsernameForm(Form):  
    username = StringField('Username')
```

Виды DSL



Высокая скорость разработки

```
import re

html_source = '...'

links = re.findall(
    pattern=r'''<a href="|'(?P<URL>.*?)\1>(?P<Text>.*?)</a>''',
    string=html_source
```

Дополнительный уровень абстракции

```
persons = select(p for p in Person if p.age > 20)[:]
```

Код могут писать "не-программисты"

```
server {
    location / {
        proxy_pass http://localhost:8080/;
    }

    location ~ \.(gif|jpg|png)$ {
        root /data/images;
    }
}
```

Проблемы и решения

- Высокая стоимость разработки DSL

Проблемы и решения

 **Достаточно один раз разобраться**

 ~~Высокая стоимость разработки DSL~~

Проблемы и решения

Достаточно один раз разобраться

- Высокая стоимость разработки DSL
- Нет специалистов со знанием языка

Проблемы и решения

Достаточно один раз разобраться

- ~~Высокая стоимость разработки DSL~~

Язык должен иметь ограниченные возможности

- ~~Нет специалистов со знанием языка~~

Проблемы и решения

Достаточно один раз разобраться

- ~~Высокая стоимость разработки DSL~~

Язык должен иметь ограниченные возможности

- ~~Нет специалистов со знанием языка~~

- Работает не для всех задач

Проблемы и решения

Достаточно один раз разобраться

- ~~Высокая стоимость разработки DSL~~

Язык должен иметь ограниченные возможности

- ~~Нет специалистов со знанием языка~~

Стоит попробовать, что бы понять

- ~~Работает не для всех задач~~

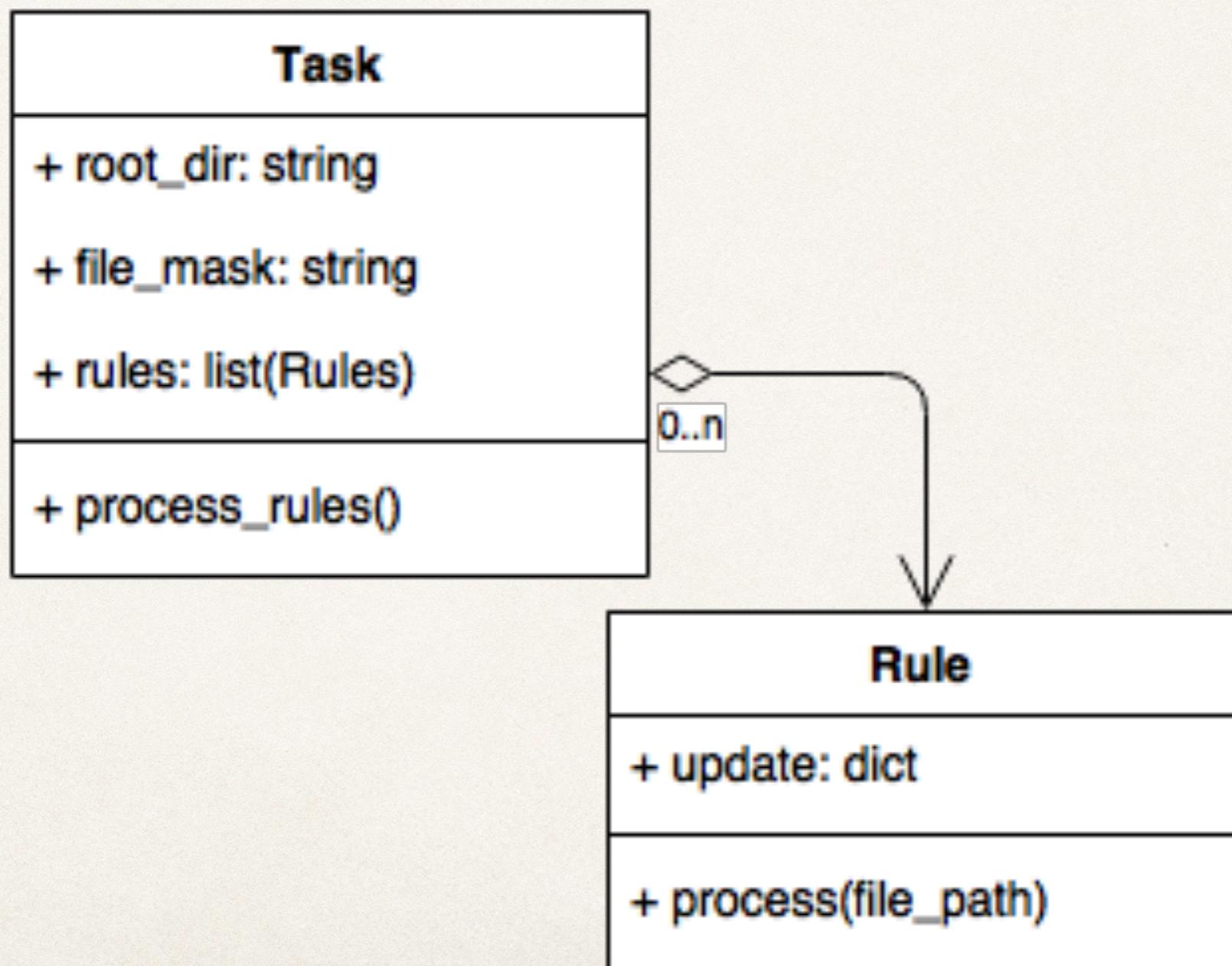
Перейдем к практике



Зачем нужна модель

- ❖ Хранит в себе всю бизнес-логику
- ❖ Позволяет использовать различные DSL
- ❖ Обеспечивает возможность тестирования

Семантическая модель



Внутренние DSL

Внутренние DSL

 Все возможности базового языка

 Ограничен базовым языком

 Привычный синтаксис

 Легко работать

Цепочки вызовов

- ❖ Все методы заполняют модель и возвращают объект
- ❖ Методы именуются исходя из смыслового контекста

```
FileUpdater()\n    .path('\'music')\n    .mask('.*metallica.*')\n    .set(Genre='Rock')\n    .set(Artist='Metallica')\n    .do()
```

Вложенные функции

- ❖ Для заполнения модели вызываются функции
- ❖ Функции именуются исходя из смыслового контекста

```
update(  
    settings(  
        path('./music'),  
        mask('.*\.\mp3')  
    ),  
    set(Artist='Metallica'),  
    set(Genre='Rock')  
)
```

Import hooks

 Очень мощный инструмент

- Вмешиваемся в работу интерпретатора
- Очень сложная отладка
- Непредсказуемые side-эффекты

Import hooks



Не используйте это в реальном мире!

```
import requests
```



```
PathFinder
```



```
FileLoader
```



```
def source_to_code(self, data, path, *, _optimize=-1):  
    ...  
    source = self.get_source(path)  
    return compile(  
        source,  
        path,  
        ...  
    )
```

my_script.py

```
WITH ".*\.mp3"
IN "../tests/music/"
SET Artist="Metallica"
SET Genre="Rock"
```

```
import internal.import_tokenizer
import examples.internal_data.my_script as script

script.task.process_rules()
```

```
import my_script as script
```



PathFinder



FileLoader



```
def source_to_code(self, data, path, *, _optimize=-1):  
    ...  
    tokens = translate(path)  
    return compile(  
        tokenize.untokenize(tokens),  
        path,  
        ...  
    )
```

```
def translate(path):
    tokens = tokenize.generate_tokens(...)
    while tokens:
        ...
        if token_value in ('IN', 'WITH'):
            ...
            yield from create_task()
        ...
        elif ...:
        else:
            yield (tok_type, value)
```

```
def translate(path):
    tokens = tokenize.generate_tokens(...)
    while tokens:
        ...
        if token_value in ('IN', 'WITH'):
            ...
            yield from create_task()
        ...
        elif ...:
        else:
            yield (tok_type, value)
```

```
def create_task():
    yield from [
        (tokenize.NAME, 'from'),
        (tokenize.NAME, 'model'),
        (tokenize.NAME, 'import'),
        (tokenize.NAME, 'Task'),
        (tokenize.OP, ','),
        (tokenize.NAME, 'Rule'),
        (tokenize.NEWLINE, '\n'),
        (tokenize.NAME, 'task'),
        (tokenize.OP, '='),
        (tokenize.NAME, 'Task'),
        (tokenize.OP, '('),
        (tokenize.OP, ')'),
        (tokenize.NEWLINE, '\n')
    ]
```

```
def create_task()  
    yield from
```

```
(t  
(t
```

```
'from',  
'model',  
'import'),  
'Task'),  
''),  
'Ru
```

```
AP,  
P, '='),  
NAME, 'T  
OP, '(:'),  
OP
```

```
]
```

No.



DSL Ruby

```
task = Task.new do
  with '*.\\.mp3'
  inside './music'
  rule do
    Artist 'Metallica'
  end
  rule do
    Genre 'Rock'
  end
end
task.run()
```

Внешние DSL

Внешние DSL

 Нет базового языка

 Сами выбираем
синтаксис

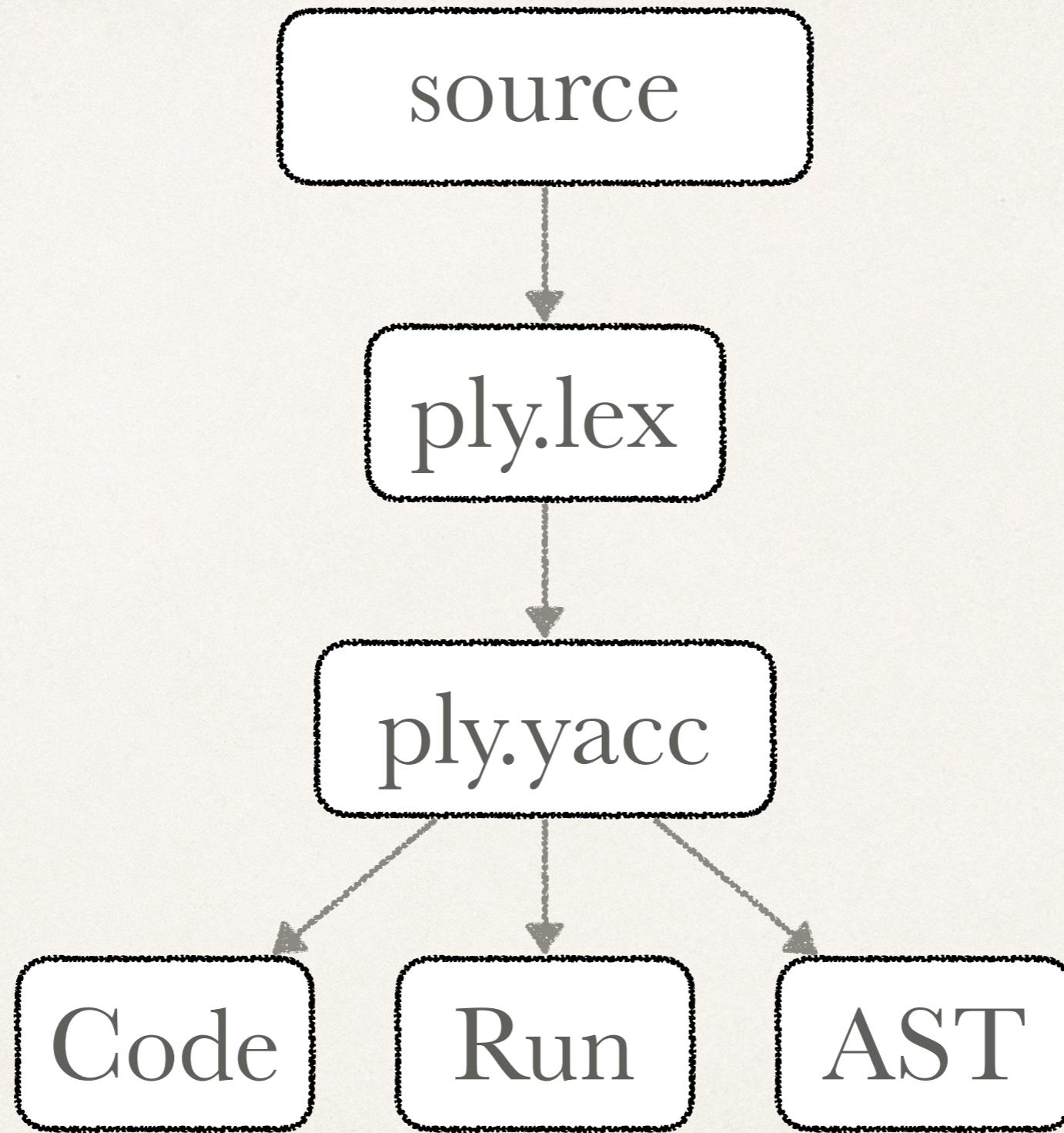
 Нет базового языка

 Необходимо
разрабатывать
анализаторы

Свой язык

```
WITH ".*\mp3"
IN "./music"
SET Artist="Metallica"
SET Genre="Rock"
```

Python Lex-Yacc (PLY)



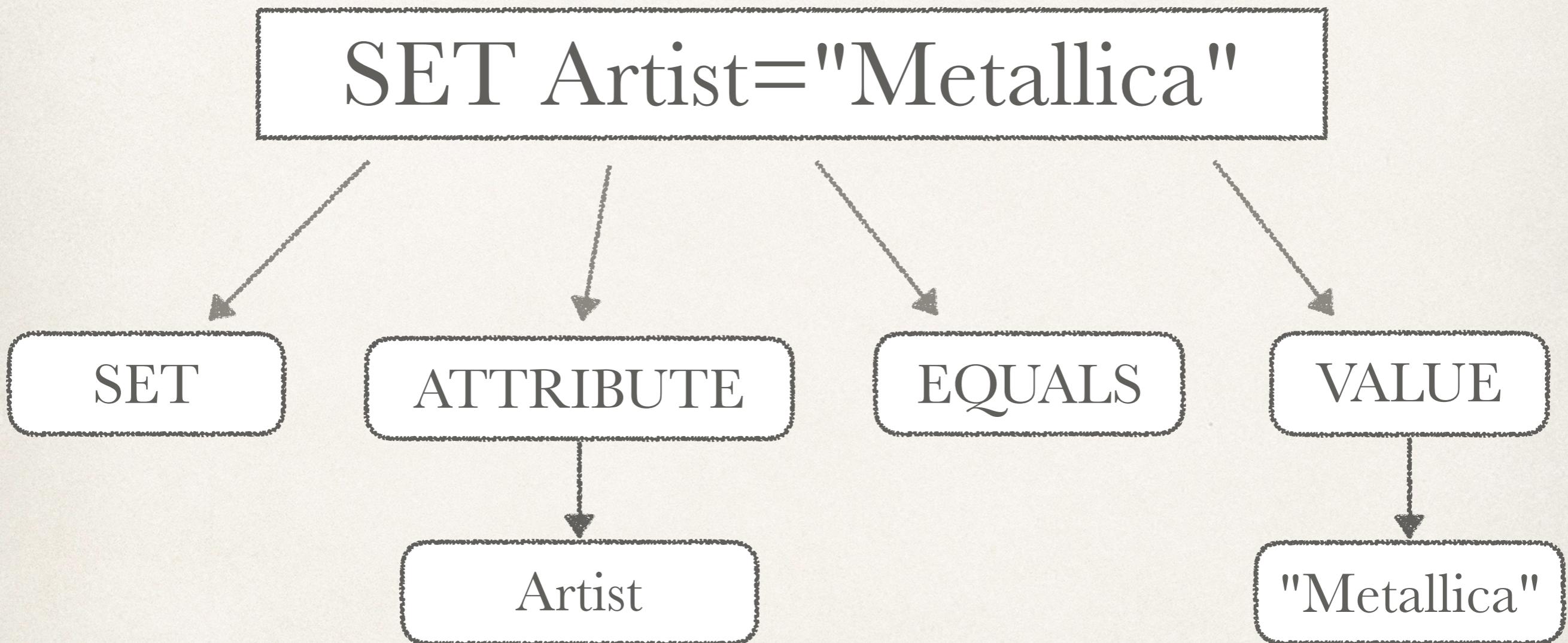
ply.lex

```
WITH ".*\mp3"  
IN "./music"  
SET Artist="Metallica"  
SET Genre="Rock"
```

Type	Value
WITH	WITH
IN	IN
SET	SET
EQUALS	=
VALUE	".*?"
ATTRIBUTE	[A-Za-z][A-Za-z0-9]*

ply.lex

```
WITH ".*\mp3"
IN "./music"
SET Artist="Metallica"
SET Genre="Rock"
```



ply.lex

WITH Artist
IN "Metallica"

```
LexToken(WITH, 'WITH', 1, 0)
LexToken(ATTRIBUTE, 'Artist', 1, 5)
LexToken(IN, 'IN', 1, 12)
LexToken(VALUE, '"Metallica"', 1, 15)
```

ply.yacc

```
WITH ".*\.mp3"
IN "./music"
SET Artist="Metallica"
SET Genre="Rock"
```

```
rule : SET ATTRIBUTE EQUALS VALUE
with : WITH VALUE
in : IN VALUE

rule_list : rule_list rule
           | rule

task : with in rule_list
      | in rule_list
```

PLY

Генерируем AST

ply.yacc

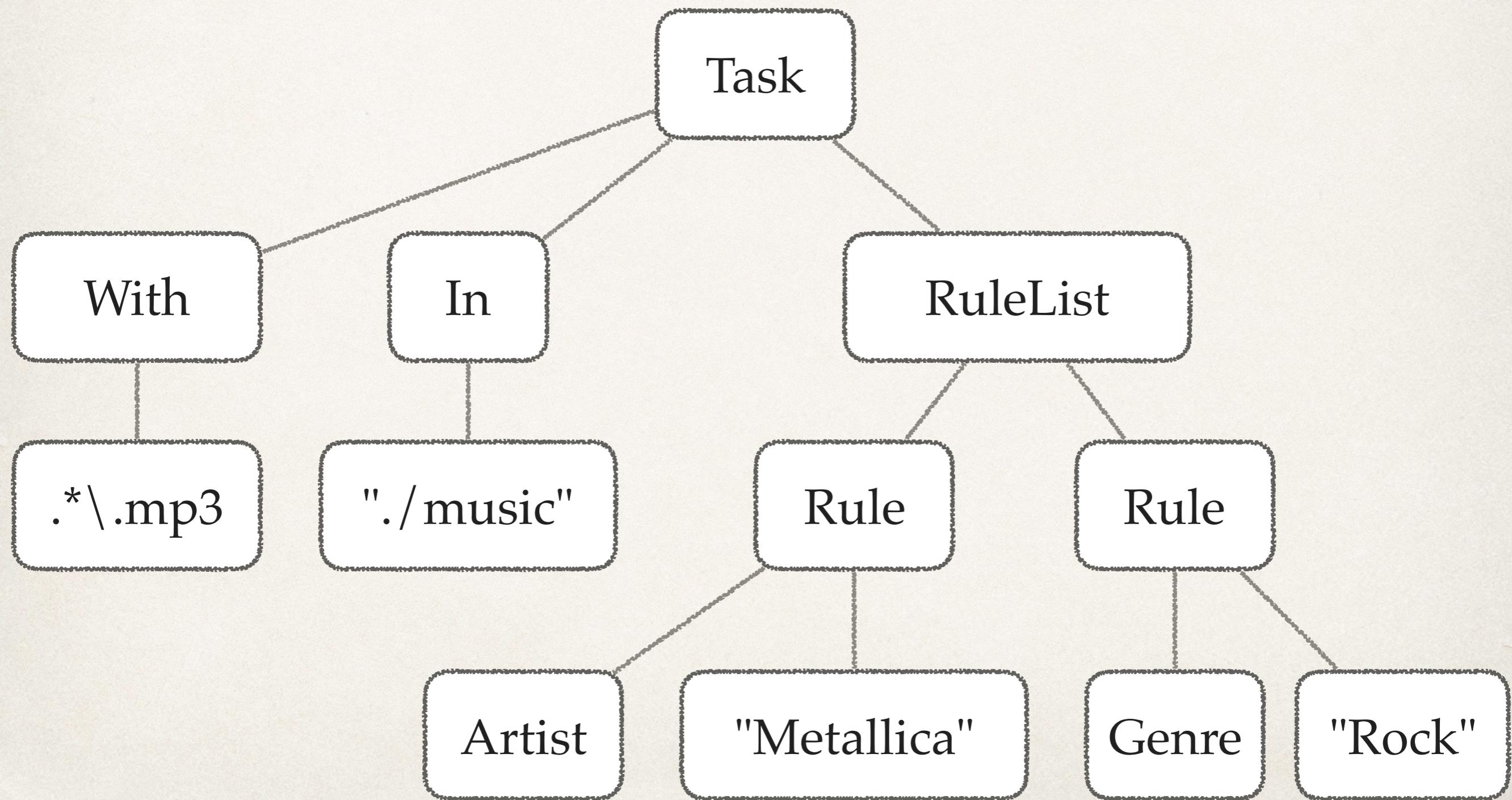
```
WITH ".*\.mp3"
IN "./music"
SET Artist="Metallica"
SET Genre="Rock"
```

```
simple_token = namedtuple(
    'simple_token', ['Name', 'Value']
)

def p_rule(self, p):
    '''rule : SET ATTRIBUTE EQUALS VALUE'''
    p[0] = simple_token(Name='RULE', Value=(p[2], p[4]))
```

ply.yacc

```
WITH ".*\mp3"
IN "./music"
SET Artist="Metallica"
SET Genre="Rock"
```



PLY

Заполняем модель

ply.yacc

```
WITH ".*\.mp3"
IN "./music"
SET Artist="Metallica"
SET Genre="Rock"
```

```
def p_rule(p):
    '''rule : SET ATTRIBUTE EQUALS VALUE'''
    p[0] = Rule(**{p[2]: p[4]})
```

ply.yacc

```
WITH ".*\mp3"
IN "./music"
SET Artist="Metallica"
SET Genre="Rock"
```

Task

```
root_dir = "./music"
file_mask = ".*\\\.mp3"
rules = [
```

Rule

```
Artist = "Metallica"
```

Rule

```
Genre = "Rock"
```

```
]
```

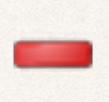
PLY

 Гибкость

 Отладка

 Обработка ошибок

 Понятный код
библиотеки

 Высокий порог входа

 Многословный

funcparserlib

funcparserlib

```
WITH ".*\mp3"
IN "./music"
SET Artist="Metallica"
SET Genre="Rock"
```

```
task = Task()

root = keyword('In') + value_of('Value') >> set_root
mask = keyword('With') + value_of('Value') >> set_mask
rule = keyword('Set') + \
    value_of('Attribute') + \
    keyword('Equals') + \
    value_of('Value') \
    >> make_rule

parser = maybe(mask) + root + many(rule)
parser.parse(source)
```

funcparserlib

```
IN "./music"  
WITH ".*\.mp3"  
SET Artist="Metallica"  
SET Genre="Rock"
```

```
get_value = lambda x: x.value  
value_of = lambda t: some(lambda x: x.type == t) >> get_value  
keyword = lambda s: skip(value_of(s))  
  
set_root = lambda value: task.set_root_dir(value[1:-1])  
set_mask = lambda value: task.set_mask(value[1:-1])  
make_rule = lambda x: task.add_rule(Rule(**{x[0]: x[1]}))
```

funcparserlib

 Компактный

 Гибкий

 Для любителей
функционального
программирования :)

- Многое приходится делать руками
- Для любителей функционального программирования :)

pyparsing

pyparsing

```
WITH ".*\.mp3"  
IN "./music"  
SET Artist="Metallica"  
SET Genre="Rock"
```

```
rule = (  
    Keyword('SET') +  
    Word(alphanums)('key') +  
    '=' +  
    QuotedString('')( 'value' )  
).setParseAction(lambda r: {r.key: r.value})
```

```
>>> rule.parseString('SET Artist="Metallica"')  
{'Artist': 'Metallica'}
```

pyparsing

```
WITH ".*\.*\mp3"
IN "./music"
SET Artist="Metallica"
SET Genre="Rock"
```

```
{
    'mask': '.*\.*\mp3',
    'root_dir': './music',
    'rules': [
        {'Artist': 'Metallica'},
        {'Genre': 'Rock'}
    ]
}
```

pyParsing

- + Понятная грамматика - Документация
- + Базовые компоненты - Отладка
- + Свои компоненты
- + Постобработка каждого элемента

Если сомневаетесь в необходимости DSL -
попробуйте

Начните с семантической модели, это
весь просто библиотека

mi.0-0.im

