The Fastest FizzBuzz *in the* West

FizzBuzz

"Write a program that prints the numbers 1 through **`n`***, however for every number divisible by 3, print* **`'fizz'`** *and for every number divisible by 5, print* **`'buzz'`***. If a number is divisible by both 3 and 5, print* **`'fizzbuzz'`***."*

Oh come on

```
>>> def fizzbuzz(n):
...     for i in range(1, n+1):
...         if i % 15 == 0:
...             print 'fizzbuzz'
...         elif i % 3 == 0:
...             print 'fizz'
...         elif i % 5 == 0:
...             print 'buzz'
...         else:
...             print i
...
```

```
>>> fizzbuzz(15)
1
2
fizz
4
buzz
fizz
7
8
fizz
buzz
11
fizz
13
14
fizzbuzz
```

NO FUN

DIVSPL

DIVSPL

(Dustin Ingram's Very Special Programming Language)

RPLY

- RPLY: https://github.com/alex/rply
  - RPython: https://en.wikipedia.org/wiki/PyPy#RPython
  - PLY: http://www.dabeaz.com/ply/
    - Lex: https://en.wikipedia.org/wiki/Lex_(software)
    - Yacc: https://en.wikipedia.org/wiki/Yacc

Let's make a lexer

Let's make a lexer

```
>>> from rply import LexerGenerator
>>> lg = LexerGenerator()
>>>
```

Let's make a lexer

```
>>> from rply import LexerGenerator
>>> lg = LexerGenerator()
>>> lg.add("ELLIPSIS", r"\.\.\.")
>>>
```

Let's make a lexer

```
>>> from rply import LexerGenerator
>>> lg = LexerGenerator()
>>> lg.add("ELLIPSIS", r"\.\.\.")
>>> lg.add("NUMBER", r"\d+")
>>>
```

Let's make a lexer

```
>>> from rply import LexerGenerator
>>> lg = LexerGenerator()
>>> lg.add("ELLIPSIS", r"\.\.\.")
>>> lg.add("NUMBER", r"\d+")
>>> lg.add("EQUALS", r"=")
>>>
```

Let's make a lexer

```
>>> from rply import LexerGenerator
>>> lg = LexerGenerator()
>>> lg.add("ELLIPSIS", r"\.\.\.")
>>> lg.add("NUMBER", r"\d+")
>>> lg.add("EQUALS", r"=")
>>> lg.add("WORD", r"[a-z]+")
>>>
```

Let's make a lexer

```
>>> from rply import LexerGenerator
>>> lg = LexerGenerator()
>>> lg.add("ELLIPSIS", r"\.\.\.")
>>> lg.add("NUMBER", r"\d+")
>>> lg.add("EQUALS", r"=")
>>> lg.add("WORD", r"[a-z]+")
>>> lg.ignore(r"\s+")  # Ignore whitespace
>>>
```

Let's make a lexer

```
>>> from rply import LexerGenerator
>>> lg = LexerGenerator()
>>> lg.add("ELLIPSIS", r"\.\.\.")
>>> lg.add("NUMBER", r"\d+")
>>> lg.add("EQUALS", r"=")
>>> lg.add("WORD", r"[a-z]+")
>>> lg.ignore(r"\s+")  # Ignore whitespace
>>> lg.ignore(r"#.*\n")  # Ignore comments
>>>
```

Let's make a lexer

```
>>> from rply import LexerGenerator
>>> lg = LexerGenerator()
>>> lg.add("ELLIPSIS", r"\.\.\.")
>>> lg.add("NUMBER", r"\d+")
>>> lg.add("EQUALS", r"=")
>>> lg.add("WORD", r"[a-z]+")
>>> lg.ignore(r"\s+")  # Ignore whitespace
>>> lg.ignore(r"#.*\n")  # Ignore comments
>>> lexer = lg.build()
>>>
```

Let's make a lexer

```
>>> iterator = lexer.lex('...foo42hut=')
>>>
```

Let's make a lexer

```
>>> iterator = lexer.lex('...foo42hut=')
>>> iterator.next()
Token('ELLIPSIS', '...')
>>>
```

Let's make a lexer

```
>>> iterator = lexer.lex('...foo42hut=')
>>> iterator.next()
Token('ELLIPSIS', '...')
>>> iterator.next()
Token('WORD', 'foo')
>>>
```

Let's make a lexer

```
>>> iterator = lexer.lex('...foo42hut=')
>>> iterator.next()
Token('ELLIPSIS', '...')
>>> iterator.next()
Token('WORD', 'foo')
>>> iterator.next()
Token('NUMBER', '42')
>>>
```

Let's make a lexer

```
>>> iterator = lexer.lex('...foo42hut=')
>>> iterator.next()
Token('ELLIPSIS', '...')
>>> iterator.next()
Token('WORD', 'foo')
>>> iterator.next()
Token('NUMBER', '42')
>>> iterator.next()
Token('WORD', 'hut')
>>>
```

Let's make a lexer

```
>>> iterator = lexer.lex('foobar!')
>>>
```

Let's make a lexer

```
>>> iterator = lexer.lex('foobar!')
>>> iterator.next()
Token('WORD', 'foobar')
>>>
```

Let's make a lexer

```
>>> iterator = lexer.lex('foobar!')
>>> iterator.next()
Token('WORD', 'foobar')
>>> iterator.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "lexer.py", line 53, in next
    raise LexingError(...)
rply.errors.LexingError
>>>
```

Let's make a parser

Let's make a parser

range $\longrightarrow$ NUMBER ELLIPSIS NUMBER

Let's make a parser

range ⟶ NUMBER ELLIPSIS NUMBER

assignment ⟶ WORD EQUALS NUMBER

Let's make a parser

range $\longrightarrow$ NUMBER ELLIPSIS NUMBER

assignment $\longrightarrow$ WORD EQUALS NUMBER

assignments $\longrightarrow$ *assignments assignment*

Let's make a parser

$$\textit{range} \longrightarrow \text{NUMBER ELLIPSIS NUMBER}$$
$$\textit{assignment} \longrightarrow \text{WORD EQUALS NUMBER}$$
$$\textit{assignments} \longrightarrow \textit{assignments assignment}$$
$$\textit{assignments} \longrightarrow \varepsilon$$

Let's make a parser

$\textit{range} \longrightarrow \text{NUMBER ELLIPSIS NUMBER}$

$\textit{assignment} \longrightarrow \text{WORD EQUALS NUMBER}$

$\textit{assignments} \longrightarrow \textit{assignments assignment}$

$\textit{assignments} \longrightarrow \varepsilon$

$\textit{main} \longrightarrow \textit{range assignments}$

Let's make a parser

```
>>> from rply import ParserGenerator
>>> pg = ParserGenerator([
...     "ELLIPSIS",
...     "EQUALS",
...     "NUMBER",
...     "WORD"
... ])
>>>
```

Let's make a parser

$\textit{range} \longrightarrow \text{NUMBER ELLIPSIS NUMBER}$

```
>>> @pg.production("range : NUMBER ELLIPSIS NUMBER")
... def range_op(p):
...     return RangeBox(int(p[0].value), int(p[2].value))
...
>>>
```

Python != Statically Typed
RPython == Statically Typed

Let's make a parser

```
>>> class RangeBox(BaseBox):
...     def __init__(self, low, high):
...         self.low = low
...         self.high = high
...     def eval(self):
...         return range(self.low, self.high + 1)
...
>>>
```

Let's make a parser

```
>>> box = RangeBox(1, 3)
<__main__.RangeBox object at 0x1046ba650>
>>>
```

Let's make a parser

```
>>> box = RangeBox(1, 3)
<__main__.RangeBox object at 0x1046ba650>
>>> box.eval()
[1, 2, 3]
>>>
```

Let's make a parser

$$\textit{assignment} \longrightarrow \text{WORD EQUALS NUMBER}$$

```
>>> @pg.production("assignment : WORD EQUALS NUMBER")
... def assignment_op(p):
...     return AssignmentBox(p[0].value, int(p[2].value))
...
>>>
```

Let's make a parser

```
>>> class AssignmentBox(BaseBox):
...     def __init__(self, word, number):
...         self.word = word
...         self.number = number
...     def eval(self, i):
...         if not i % int(self.number):
...             return self.word
...         return ''
>>>
```

Let's make a parser

```
>>> box = AssignmentBox('foo', 7)
>>> box.eval(40)
''
>>> box.eval(42)
'foo'
>>>
```

Let's make a parser

$assignments \longrightarrow assignments\ assignment$

$assignments \longrightarrow \varepsilon$

```
>>> @pg.production("assignments : assignments assignment")
... @pg.production("assignments : ")
... def expr_assignments(p):
...     if p:
...         return p[0] + [p[1]]
...     return []
...
>>>
```

Let's make a parser

$main \longrightarrow range\ assignments$

```
>>> @pg.production("main : range assignments")
... def main(p):
...     return ProgramBox(p[0], p[1])
...
>>>
```

Let's make a parser

```
>>> class ProgramBox(BaseBox):
...     def __init__(self, range_box, assignment_boxes):
...         self.range_box = range_box
...         self.assignment_boxes = assignment_boxes
...
```

Let's make a parser

```
...     def eval(self):
...         return "\n".join(
...             "".join(
...               assignment.eval(i)
...               for assignment in self.assignment_boxes
...             ) or str(i)
...             for i in self.range_box.eval()
...         ) + "\n"
...
>>>
```

Let's make a parser

```
>>> parser = pg.build()
```

Let's make an interpreter

Let's make an interpreter

```
>>> def main():
...     if len(sys.argv) > 1:
...         with open(sys.argv[1], 'r') as f:
...             result = parser.parse(lexer.lex(f.read()))
...             sys.stdout.write(result.eval())
...     else:
...         sys.stdout.write("Please provide a filename.")
...
>>>
```

LIVECODING!