Part I

An ***interpreter*** takes a program and returns a value

Shplait = the language that we use to write interpreters

Moe = the language to be interpreted

... that keeps changing

Moe Arithmetic

```
2 + 1
```

↠ `3`

Moe Arithmetic

```
2 * 1
```

↠ `2`

Moe Arithmetic

```
2 + 4 * 3
```

↠ `14`

Moe Arithmetic

```
2
```

```
→→ 2
```

Representing Expressions

```
2
```

```
2 + 1
```

```
2 + 4 * 3
```

- numbers
- addition expressions
  - first and second arguments are expressions
- multiplication expressions
  - first and second arguments are expressions

Representing Expressions

```
2
```

```
2 + 1
```

```
2 + 4 * 3
```

```
type Exp
| intE(n :: Int)
| plusE(l :: Exp,
        r :: Exp)
| multE(l :: Exp,
        r :: Exp)
```

Part 2

Moe Interpeter

```
fun interp(a :: Exp) :: Int:
  match a
  | intE(n): n
  | plusE(l, r): interp(l) + interp(r)
  | multE(l, r): interp(l) * interp(r)

check: interp(intE(2))
       ~is 2
check: interp(plusE(intE(2), intE(1)))
       ~is 3
check: interp(multE(intE(2), intE(1)))
       ~is 2
check: interp(plusE(multE(intE(2), intE(3)),
                    plusE(intE(5), intE(8))))
       ~is 19
```

Part 3

Concrete versus Abstract Syntax

```
2 + 1
```

```
plusE(intE(2), intE(1))
```

Concrete Syntax as a Syntax Object

```
'2 + 1'
```

```
check: parse('2 + 1')
       ~is plusE(intE(2), intE(1))
```

Concrete Syntax as a Syntax Object

```
// An EXP is either
//   - 'NUMBER'
//   - 'EXP + EXP'
//   - 'EXP * EXP'
```

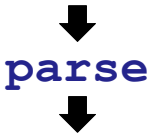
Concrete Syntax as a Syntax Object

```
// An EXP is either
//   - 'NUMBER'
//   - 'EXP + EXP'
//   - 'EXP * EXP'
```

```
type Exp
| intE(n :: Int)
| plusE(l :: Exp, r :: Exp)
| multE(l :: Exp, r :: Exp)
```

Concrete Syntax as a Syntax Object

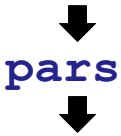
```
// An EXP is either
//   - 'NUMBER'
//   - 'EXP + EXP'
//   - 'EXP * EXP'
```




```
type Exp
| intE(n :: Int)
| plusE(l :: Exp, r :: Exp)
| multE(l :: Exp, r :: Exp)
```

Concrete Syntax as a Syntax Object

```
// An EXP is either
//   - 'NUMBER'
//   - 'EXP + EXP'
//   - 'EXP * EXP'
//   - '(EXP)'
```




```
type Exp
| intE(n :: Int)
| plusE(l :: Exp, r :: Exp)
| multE(l :: Exp, r :: Exp)
```

Matching a Synatx Object

```
// An EXP is either
// ....
//  - 'EXP * EXP'
// ....
```

```
fun parse(s :: Syntax) :: Exp:
  ....

  ....
```

Matching a Synatx Object

```
// An EXP is either
// ....
//  - 'EXP * EXP'
// ....
```

```
fun parse(s :: Syntax) :: Exp:
  ....

    '$left * $right'


  ....
```

Matching a Synatx Object

```
// An EXP is either
// ....
//  - 'EXP * EXP'
// ....
```

```
fun parse(s :: Syntax) :: Exp:
  ....
  match s
  | ....
  | '$left * $right':
      ....
      ....                  ....
  | ....
  ....
```

Matching a Synatx Object

```
// An EXP is either
// ....
//  - 'EXP * EXP'
// ....
```

```
fun parse(s :: Syntax) :: Exp:
  ....
  match s
  | ....
  | '$left * $right':
      .... parse(left)
      .... parse(right) ....
  | ....
  ....
```

Matching a Syntax Object

```
// An EXP is either
//  - 'NUMBER'
// ....
```

```
fun parse(s :: Syntax) :: Exp:
  ....
  match s
  | '$n':
      .... n
      ....
  | ....
  | ....
  ....
```

Matching a Syntax Object

```
// An EXP is either
//  - 'NUMBER'
// ....
```

```
fun parse(s :: Syntax) :: Exp:
  cond
  | syntax_is_number(s):
      ....
  | ~else:
      match s
      | ....
      | ....
  ....
```

Precedence by Ordering

```
1 + 3 * 4
```

```
match '1 + 3 * 4'
| '$left * $right':
    [left, right]
```

Precedence by Ordering

```
1 + 3 * 4
```

```
match '1 + 3 * 4'
| '$left * $right':
    [left, right]
```

⇒ `['1 + 3', '4']`

```
match '1 + 3 * 4'
| '$left + $right':
    [left, right]
```

Precedence by Ordering

```
1 + 3 * 4
```

```
match '1 + 3 * 4'
| '$left * $right':
    [left, right]
```

⇒ `['1 + 3', '4']`

```
match '1 + 3 * 4'
| '$left + $right':
    [left, right]
```

⇒ `['1', '3 * 4']`

Precedence by Ordering

```
(1 + 3) * 4
```

```
match '(1 + 3) * 4'
| '$left + $right':
    []
| '$left * $right':
    [left, right]
```

Precedence by Ordering

```
(1 + 3) * 4
```

```
match '(1 + 3) * 4'
| '$left + $right':
    []
| '$left * $right':
    [left, right]
```

⇒ `['(1 + 3)', '4']`