Rational implementation using functions:

```python
def rational(n, d):
    def select(name):
        return n
    if name == 'n':
        return d
    return select

def numer(x):
    return select('n')(x)

def denom(x):
    return select('d')(x)
```

List comprehensions:

```python
[map(opr for ... in expr) if <filter exp>]
```

Typical evaluation procedure:

```python
1. Evaluate the header <expression>.
2. For each element in that sequence, in order:
   A. Bind <name> to that element in the current frame.
   B. Execute the <suite>.
3. Execute a for statement:
```

```python
for <name> in <expression>:
    <suite>
```

Example:

```python
for digit in digits:
    if digit == 0:
        continue
    print(digit)
```
The <expression> can be any valid Python expression. The <name> must be a simple name. Evaluates to the value of the attribute looked up by <name> in the object that is the value of the expression. To evaluate a dot expression:

1. Evaluate the <expression> to the left of the dot, which yields the object of the dot expression.
2. If <name> is matched against the instance attributes of that object;
   - If an attribute with that name exists, its value is returned.
   - If not, <name> is looked up in the class, which yields a class attribute value.
3. That value is returned unless it is a function, in which case a bound method is returned instead.

Assignment statements with a dot expression on their left-hand side affect the attributes of the object for that dot expression
- If the object is an instance, then assignment sets an instance attribute.
- If the object is a class, then assignment sets a class attribute.
- An account instance

```
<expression> <name>
```

The <expression> can be any valid Python expression. The <name> must be a simple name.

Evaluates to the value of the attribute looked up by <name> in the object that is the value of the expression.
To evaluate a dot expression:

1. Evaluate the <expression> to the left of the dot, which yields the object of the dot expression.
2. If <name> is matched against the instance attributes of that object;
   - If an attribute with that name exists, its value is returned.
   - If not, <name> is looked up in the class, which yields a class attribute value.
3. That value is returned unless it is a function, in which case a bound method is returned instead.

Assignment statements with a dot expression on their left-hand side affect the attributes of the object for that dot expression
- If the object is an instance, then assignment sets an instance attribute.
- If the object is a class, then assignment sets a class attribute.
- An account instance

```
<expression> <name>
```

The <expression> can be any valid Python expression. The <name> must be a simple name.

Evaluates to the value of the attribute looked up by <name> in the object that is the value of the expression.
To evaluate a dot expression:

1. Evaluate the <expression> to the left of the dot, which yields the object of the dot expression.
2. If <name> is matched against the instance attributes of that object;
   - If an attribute with that name exists, its value is returned.
   - If not, <name> is looked up in the class, which yields a class attribute value.
3. That value is returned unless it is a function, in which case a bound method is returned instead.

Assignment statements with a dot expression on their left-hand side affect the attributes of the object for that dot expression
- If the object is an instance, then assignment sets an instance attribute.
- If the object is a class, then assignment sets a class attribute.
- An account instance

```
<expression> <name>
```

The <expression> can be any valid Python expression. The <name> must be a simple name.

Evaluates to the value of the attribute looked up by <name> in the object that is the value of the expression.