Homework 4

The assignment is to implement predicate-calculus expressions and simplification (beta-reduction), as described on Handout 7.

Be sure that your functions behave as indicated in the examples.

1. Variables

a. Define the class `Variable`. The only method is `__repr__()`. Example:

```
>>> from hw4 import *
>>> Variable('x')
x
```

b. Define the function `fresh_variable()`. It uses a global variable to keep track of which number to use next. Example:

```
>>> v = fresh_variable()
>>> v
_1
>>> isinstance(v, Variable)
True
>>> fresh_variable()
_2
```

2. Expressions

a. Define class `Expr`. The only method is `__repr__()`. Example:

```
>>> e = Expr(['lambda',
...           Variable('x'),
...           Expr(['chases', 'Fido', Variable('x')])])
...
>>> e
(lambda x (chases Fido x))
>>> isinstance(e, Expr)
True
>>> len(e[2])
3
>>> e[0]
'lambda'
>>> isinstance(e[2], Expr)
True
>>> e[2][1]
'Fido'
```

b. Define function `is_variable_name(s)`. It takes a string and returns `True` just in case the string looks like a variable: exactly one letter, optionally followed by any number of digits.

```
>>> bool(is_variable_name('x'))
True
>>> bool(is_variable_name('X'))
True
>>> bool(is_variable_name('x12'))
True
>>> bool(is_variable_name('2d'))
False
>>> bool(is_variable_name('cat'))
False
```

c. Define function `parse_expr(s)`. It takes a string and returns an expression. An atomic expression is either a string (representing a constant) or a `Variable`, and a complex expression is an `Expr` whose elements are expressions. Use `is_variable_name()` to distinguish variables from constants.

```
>>> e = parse_expr('(lambda x (chases Fido x))')
>>> e
(lambda x (chases Fido x))
>>> e[0]
'lambda'
>>> e[1]
x
>>> isinstance(e[0], Variable)
False
>>> isinstance(e[1], Variable)
True
```

3. Simplification

a. Define the function `is_lambda_expr(e)`.

```
>>> E = parse_expr
>>> is_lambda_expr(E('(lambda (x y) (foo y x))'))
True
>>> is_lambda_expr(E('((lambda x x) foo)'))
False
```

b. Define the function `normalize(e)`.

```
>>> e = normalize(E('((lambda x ((lambda x (bar x)) x)) x)'))
>>> e
((lambda (_3) ((lambda (_4) (bar _4)) _3)) x)
```

c. Define the function `simplify(e)`.

```
>>> simplify(e)
(bar x)
```

4. Make sure you get the right results for these additional tests. (Note that we set `E = parse_expr` above.)

a. Warm-up

```
>>> e = E('(all x (if (dog x) (barks x)))')
>>> e[1]
x
>>> isinstance(e[1], Variable)
True
>>> e[2][1][0]
'dog'
```

b. Normalization and simplification

```
>>> e = E('((lambda x ((lambda x (foo x)) x)) x)')
>>> e
((lambda x ((lambda x (foo x)) x)) x)
>>> normalize(e)
((lambda (_7) ((lambda (_8) (foo _8)) _7)) x)
>>> simplify(e)
(foo x)
>>> e = E('''((lambda (x y) (foo (bar y) x))
...           (mother jack)
...           (father jill))''')
>>> e
((lambda (x y) (foo (bar y) x)) (mother jack) (father jill))
>>> simplify(e)
(foo (bar (father jill)) (mother jack))
>>> e = E('((lambda (x f) (f x)) fido (lambda x x))')
>>> e
((lambda (x f) (f x)) fido (lambda x x))
>>> simplify(e)
'fido'
```

c. From the Handout:

```
>>> simplify(E('((lambda x x) fido)'))
'fido'
>>> simplify(E('((lambda f (f fido)) (lambda x (dog x)))'))
(dog fido)
>>> simplify(E('''(((lambda f (lambda x (f x x)))
...                 (lambda (x y) (likes y x)))
...                fido)'''))
(likes fido fido)
```