Lambda Expressions in Python

So far, each time we have wanted to define a new function, we needed to give it a name. But for other types of expressions, we don't need to associate intermediate values with a name. That is, we can compute `a*b + c*d` without having to name the subexpressions `a*b` or `c*d`, or the full expression. In Python, we can create function values on the fly using `lambda` expressions, which evaluate to unnamed functions. A lambda expression evaluates to a function that has a single return expression as its body. Assignment and control statements are not allowed.

Lambda expressions are limited: They are only useful for simple, one-line functions that evaluate and return a single expression. In those special cases where they apply, lambda expressions can be quite expressive.

```
>>> def compose1(f, g):
        return lambda x: f(g(x))
```

We can understand the structure of a `lambda` expression by constructing a corresponding English sentence:

```
     lambda            x            :          f(g(x))
"A function that    takes x    and returns     f(g(x))"
```

The result of a lambda expression is called a lambda function. It has no intrinsic name (and so Python prints `<lambda>` for the name), but otherwise behaves like any other function.

```
>>> s = lambda x: x * x
>>> s
<function <lambda> at 0xf3f490>
>>> s(12)
144
```

In an environment diagram, the result of a lambda expression is a function as well, named with the greek letter λ (lambda). Our compose example can be expressed quite compactly with lambda expressions.

```
1  def compose1(f, g):
2      return lambda x: f(g(x))
3
4  f = compose1(lambda x: x * x, lambda x: x + 1)
5  result = f(12)
```

Some programmers find that using unnamed functions from lambda expressions to be shorter and more direct. However, compound `lambda` expressions are notoriously illegible, despite their brevity. The following definition is correct, but many programmers have trouble understanding it quickly.

```
>>> compose1 = lambda f,g: lambda x: f(g(x))
```

In general, Python style prefers explicit `def` statements to lambda expressions, but allows them in cases where a simple function is needed as an argument or return value.

Such stylistic rules are merely guidelines; you can program any way you wish. However, as you write programs, think about the audience of people who might read your program one day. When you can make your program easier to understand, you do those people a favor.

The term *lambda* is a historical accident resulting from the incompatibility of written mathematical notation and the constraints of early type-setting systems.

It may seem perverse to use lambda to introduce a procedure/function. The notation goes back to Alonzo Church, who in the 1930's started with a "hat" symbol; he wrote the square function as "ŷ . y × y". But frustrated typographers moved the hat to the left of the parameter and changed it to a capital lambda: "Λy . y × y"; from there the capital

lambda was changed to lowercase, and now we see "λy . y × y" in math books and `(lambda (y) (* y y))` in Lisp.

—Peter Norvig (norvig.com/lispy2.html)

Despite their unusual etymology, `lambda` expressions and the corresponding formal language for function application, the *lambda calculus*, are fundamental computer science concepts shared far beyond the Python programming community

Source : http://inst.eecs.berkeley.edu/~cs61A/book/chapters/functions.html#lambda-expressions