

Lambdas

Surya Duggirala

January 2017

1 Overview

Lambdas are a useful feature in Python and some other languages (notable newer versions of Java). It's helpful in that you can place it wherever you want without explicitly defining a new function. For example, imagine some function that takes in two parameters. One parameter is an integer and the other parameter is a function that operates on that integer.

```
>>> def func(num, f):  
...     return f(num)  
...  
>>>
```

So we've established a very basic function that will return the value of the function f operating on number num . We could define a fully fledged function but what if we're just running some basic functions that don't really require a ton of lines to write. Why bother writing a full out function when we can just use a **lambda** statement?

```
>>> func(4, lambda x: x + 1)  
5  
>>>
```

See how simple and mess free that was? Lambdas as a general rule are just anonymous functions that we can use once and then forget about for the rest of our lives. A word of advice, I'm not a fan of studying material in terms of an exam but it's important to note that in the algorithm portions of midterms and finals in CS61A lambdas are thrown in occasionally to shake things up. If you think you weren't given enough lines for your implementation then think of lambdas. There's a decent chance that they may be what you need.

2 Understanding the syntax

A lambda function's structure is similar to that of our regular run of the mill python function. Let's compare them.

```

>>> def func(x, y, z):
...     return x + y + z
...
>>> func(1, 2, 3)
6
>>> lambda_func = lambda x, y, z: x + y + z
>>> lambda_fun(1, 2, 3)
6
>>>

```

See how concise lambda functions are? It's important to remember that regardless of how confusing the syntax can be at times, they behave the **exact** same way as regular functions. They can be assigned to variables and they behave the same way on environment diagrams. A key difference between ordinary functions and lambdas is that lambdas **must** have a return value. If you're a little shaky on that concept, read through my notes on return values which should clear up a lot of doubts.

3 What would Python Print: Lambdas

So let's look at some examples of how python expresses lambdas.

```

>>> lam = lambda x, y, z: print(x) print(y) print(z)
File "<stdin>", line 1
    lambda x, y, z: print(x) print(y) print(z)
                        ^

```

```

SyntaxError: invalid syntax
>>> # Remember that lambdas must have a return value and
>>> # that they can only execute one line of code.
>>> lam = lambda x: True
>>> lam(4)
True
>>> lam(False)
True
>>> lam(lambda y: False)
True
>>> # Don't lose track of what the function is doing.
>>> # This function is no different than the following.
>>> def func(x):
...     return True
...
>>> func(False)
True
>>> # This function and the lambda preceding it will
>>> # always return True.
>>> lam2 = lambda x: y

```

```

>>> lam2(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 1, in <lambda>
NameError: global name 'y' is not defined
>>> # This one should be pretty obvious. We never defined
>>> # a variable y so there isn't a logical answer
>>> # to this.
>>> # Think about this function.
>>> def func(x):
...     return y
...
>>> # Makes sense?
>>> # Let's try something interesting now.
>>> lam3 = lambda x: if x == 4: True
File "<stdin>", line 1
    lam3 = lambda x: if x == 1: True
                        ^
SyntaxError: invalid syntax
>>> # Remember that lambdas run the bare minimum
>>> # kind of instruction

```