lambda expressions

One of Pythons most useful (and for beginners, confusing) tools is the lambda expression. lambda expressions allow us to create "anonymous" functions. This basically means we can quickly make ad-hoc functions without needing to properly define a function using def.

Function objects returned by running lambda expressions work exactly the same as those created and assigned by defs. There is key difference that makes lambda useful in specialized roles:

lambda's body is a single expression, not a block of statements.

- The lambda's body is similar to what we would put in a def body's return statement. We simply type the result as an expression instead of explicitly returning it. Because it is limited to an expression, a lambda is less general that a def. We can only squeeze design, to limit program nesting. lambda is designed for coding simple functions, and def handles the larger and more complicated tasks.

Lets slowly break down a lambda expression by deconstructing a function:

In [ ]:
```
def square(num):
    result = num**2
    return result
```

In [2]:
```
square(2)
```

Out[2]: 4

Continuing the breakdown:

In [3]:
```
def square(num):
    return num**2
```

In [4]:
```
square(2)
```

Out[4]: 4

We can actually write this in one line (although it would be bad style to do so)

In [5]:
```
def square(num): return num**2
```

In [6]:
```
square(2)
```

Out[6]: 4

This is the form a function that a lambda expression intends to replicate. A lambda expression

can then be written as:

In [7]:
```
lambda num: num**2
```

Out[7]:
```
<function __main__.<lambda>>
```

Note how we get a function back. We can assign this function to a label:

In [8]:
```
square = lambda num: num**2
```

In [9]:
```
square(2)
```

Out[9]:
```
4
```

And there you have it! The breakdown of a function into a lambda expression! Lets see a few more examples:

Example 1

Check if a number is even

In [13]:
```
even = lambda x: x%2==0
```

In [14]:
```
even(3)
```

Out[14]:
```
False
```

In [15]:
```
even(4)
```

Out[15]:
```
True
```

Example 2

Grab first character of a string:

In [22]:
```
first = lambda s: s[0]
```

In [23]:
```
first('hello')
```

Out[23]:
```
'h'
```

Example 3

Reverse a string:

In [24]:
```
rev = lambda s: s[::-1]
```

In [25]:
```
rev('hello')
```

Out[25]:
```
'olleh'
```

Example 4

Just like a normal function, we can accept more than one function into a lambda expression:

In [17]:
```
adder = lambda x,y : x+y
```

In [19]:
```
adder(2,3)
```

Out[19]:
```
5
```

lambda expressions really shine when used in conjunction with map(),filter() and reduce(). Each of those functions has its own lecture, so feel free to explore them if you're very interested in lambda.

I highly recommend reading this blog post at Python Conquers the Universe (https://pythonconquerstheuniverse.wordpress.com/2011/08/29/lambda_tutorial/) for a great breakdown on lambda expressions and some explanations of common confusions!