Lambda Functions

(anonymous functions) from Lisp & functional programming

In [2]:
```
tmp = lambda x: x**2
print  type(tmp)
```

```
<type 'function'>
```

In [3]:
```
tmp(2)
```

Out[3]: 4

In [9]:
```
# forget about creating a new function name...just do it!
(lambda x,y: x**2+y)(2,4.5)
```

Out[9]: 8.5

In [10]:
```
## create a list of lambda functions
lamfun = [lambda x: x**2, lambda x: x**3, \
          lambda y: math.sqrt(y) if y >= 0 else "Really? I mean really? %f" % y]
```

In [11]:
```
for l in lamfun: print l(-1.3)
```

```
1.69
-2.197
Really? I mean really? -1.300000
```

lambda functions are meant to be short, one liners. If you need more complex functions, probably better just to name them

[back]

Filter is a certain way to do list comprehension

filter(function, sequence)" returns a sequence consisting of those items from the sequence for which function(item) is true

In [14]:
```
mylist=[num for num in range(101) if (num & 2) and (num & 1) and (num % 11 != 0.0)]
print mylist
```

```
[3, 7, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 59, 63, 67, 71, 75, 79, 83, 87, 91,
95]
```

In [16]:
```
def f(num): return (num & 2) and (num & 1) and (num % 11 != 0.0)
mylist = filter(f,xrange(101))
print mylist
```

```
[3, 7, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 59, 63, 67, 71, 75, 79, 83, 87, 91,
95]
```

if the input is a string, so is the output...

In [19]:
```
## also works on strings...try it with lambdas!
import string
a="Charlie Brown said \"!@!@$@!\""
filter(lambda c: c in string.ascii_letters,a)
```

Out[19]: 'CharlieBrownsaid'

In [20]:
```
filter(lambda num: (num & 2) and (num & 1) and (num % 11 != 0.0),xrange(101))
```

Out[20]:
```
[3,
 7,
 15,
 19,
 23,
 27,
 31,
 35,
 39,
 43,
 47,
 51,
 59,
 63,
 67,
 71,
 75,
 79,
 83,
 87,
 91,
 95]
```

[back]

Map is just another way to do list comprehension

In [21]:
```
def cube_it(x): return x**3
map(cube_it,xrange(1,10))
```

Out[21]:
```
[1, 8, 27, 64, 125, 216, 343, 512, 729]
```

In [22]:
```
map(lambda x: x**3, xrange(1,10))
```

Out[22]:
```
[1, 8, 27, 64, 125, 216, 343, 512, 729]
```

Reduce returns one value

reduce(function, sequence) returns a single value constructed by calling the binary function function on the first two items of the sequence, then on the result and the next item, and so on

In [23]:
```
# sum from 1 to 10
reduce(lambda x,y: x + y, xrange(1,11))
%timeit reduce(lambda x,y: x + y, xrange(1,11))
```

```
1000000 loops, best of 3: 1.56 us per loop
```

In [24]:
```
# sum() is a built in function...it's bound to be faster
%timeit sum(xrange(1,11))
```

```
1000000 loops, best of 3: 478 ns per loop
```

zip()

built in function to pairwise concatenate items in iterables into a list of tuples

In [25]:
```
zip(["I","you","them"],["=spam","=eggs","=dark knights"])
```

Out[25]:
```
[('I', '=spam'), ('you', '=eggs'), ('them', '=dark knights')]
```

In [26]:
```
zip(["I","you","them"],["=spam","=eggs","=dark knights"],["!","?","#"])
```

Out[26]:
```
[('I', '=spam', '!'), ('you', '=eggs', '?'), ('them', '=dark knights', '#')]
```

In [27]:
```
zip(["I","you","them"],["=spam","=eggs","=dark knights"],["!","?"])
```

Out[27]:
```
[('I', '=spam', '!'), ('you', '=eggs', '?')]
```

In [30]:
```
questions = ['name', 'quest', 'favorite color']
answers = ['lancelot', 'the holy grail', 'blue']
for q, a in zip(questions, answers):
  print 'What is your %s?  It is %s.' % (q, a)
```

```
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

not to be confused with `zipfile` module which exposes file compression

[back]

In [ ]:
```

```