Advanced functional programming

Reuven M. Lerner, PhD
reuven@lerner.co.il

min() and max()

- These two functions, by default, return the largest or smallest element of a sequence
- But you can also pass a function as an (optional) second parameter — letting you get the largest or smallest element of a sequence, regardless of the contents

map, filter, reduce

- Mainstays of functional programming
- Modify the data, but don't set state
- Chain them together for maximal effect
- Use anonymous functions
- List comprehensions are replacing these to a large degree

map

- Takes a sequence as input
- Produces a sequence (of equal length) as output
- Function transforms the sequence
- Like the left side of a list comprehension

map examples

```
mylist = [1,2,3,4,5]

map(lambda x: x*x, mylist)

map(lambda x: x*x, map(lambda x: x*x,
mylist))

map(square, mylist)
```

filter

- filter keeps only those elements for which the function returns True:

```
tup = (9, 15, 200, 2, 3, 80)

filter(lambda x: x%10, tup)

filter(lambda x: False, tup)
```

Primes with filter

```
nums = range(2, 50)

for i in range(2, 8):

    nums = filter(lambda x: x == i or x
    % i, nums)
```

reduce

- Gives us a single value from a sequence
- lambda takes two arguments here — an accumulated value, and an iterated value

```
reduce(lambda total, current: total + current, t)
```

- or we can just say sum(t) !

Nested list comprehensions!

- A typical example:

```
[(x,y) for x in range(5) for y in range(5)]
```

- Huh?!?

More readable

```
[(x,y)
 for x in range(5)
 for y in range(5)]
```

More readable

```
[(x,y)
 for x in range(5)
 for y in range(5)]
```

More sophistication

```
[(x,y)
 for x in range(5)
 for y in range(x+1)]
```

Game scores

```
{'Reuven':[300, 250, 350, 400],
 'Atara':[200, 300, 450, 150],
 'Shikma':[250, 380, 420, 120],
 'Amotz':[100, 120, 150, 180]
}

def average(scores):
 return sum(scores) / len(scores)
```

Get all game scores

```
>>> [score
    for score_list in s.values()
    for score in score_list]
```

```
[300, 250, 350, 400, 100, 120, 150, 180,
200, 300, 450, 150, 250, 380, 420, 120]
```

Average score across all people

```
>>> average([ one_score
 for one_player_scores in scores.values()
 for one_score in one_player_scores ])
```

Average score across all people (but ignoring <200)

```
>>> [ one_score
    for one_player_scores in scores.values()
    for one_score in one_player_scores
 if one_score > 200]


[300, 250, 350, 400, 300, 450, 250, 380, 420]
```

Rooms

```
rooms = [[
 {'age': 14, 'hobby': 'horses', 'name': 'A'},
 {'age': 12, 'hobby': 'piano', 'name': 'B'},
 {'age': 9, 'hobby': 'chess', 'name': 'C'}],
 [{'age': 15, 'hobby': 'programming', 'name': 'D'},
 {'age': 17, 'hobby': 'driving', 'name': 'E'}],
 [{'age': 45, 'hobby': 'writing', 'name': 'F'},
 {'age': 43, 'hobby': 'chess', 'name': 'G'}]]
```

Names of guests

```
>>> [ person['name']
    for room in rooms
    for person in room ]


['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

Chess players' names

```
>>> [ person['name']

 for room in rooms

 for person in room

 if person['hobby'] == 'chess' ]


['C', 'G']
```