3 Functional model

Key terms: `lambda zip map filter`

Reading: A. M. Kuchling's Functional Programming HOWTO[9]
Reading: Carlos Balderas's Iterating With Python Lambdas[10]

Exercise: Use `map` and `filter` with `lambda` functions to identify the users whose shell access is disabled in /etc/passwd.

Python encompasses all of procedural, object oriented, and functional programming. Since many programmers know how to define functions without programming functionally, we here devote some attention to this valuable model. A wholly functional program has only one statement, and therefore no loops or conditional blocks, which increase complexity and tend to introduce bugs. You won't need to write purely functional programs to benefit from this structure. For example, we can define conditional branches using inline ternary expressions:

```
# Print the number of arguments or remark on their absence:

print ("No arguments" if len(sys.argv)<=1 else str(len(sys.argv)))
```

Functions appear in places you might not already expect. Python syntax implicitly refers to operations that are functions with English names. The language's basic operators actually refer to functions: `+` is `operator.add()`, `!` is `operator.invert()`, and so on. Sort orders are defined by functions too: `sorted(dir(),key=len`. New anonymous functions can be defined inline; the keyword to do so is `lambda`.

```
# Demonstrate inline function definition:

cube = lambda x: x**3
print(cube(5))
```

Generally, data can be translated from one form to another; it can be winnowed; and it can be summarized. Mappings, filtrations, and reductions can solve many if not all problems. We have been doing these things all along with explicit loops, which don't exist in the functional paradigm.

Comprehensions consist of a filtration and a mapping, i.e., calling `filter()` and `map()`. A filtration function returns an iterable including perhaps fewer than all the input elements; a mapping function returns some object for every element of an iterable. Using and combining functions makes it crucial to track the nested pairs of parentheses; you may want to indent their contents as a guide.

```
# Visualize integer multiples of three, up to twenty:
# The arrangement of the parentheses is debatable.

print(
 sep=' ',
 *map(lambda n: '.'*n,
  filter(lambda n: not n%3,
   range(20))))
```

`reduce`, in the `functools` package, applies another function iteratively to collapse a collection into a single value. Common specifications of reductions (`len()`, `max`, `any`, `in`, etc.) are, of course, built-in.

Next week, we will examine generators and lazy evaluation.

[9] `https://docs.python.org/3.5/howto/functional.html`
[10] `https://caisbalderas.com/blog/iterating-with-python-lambdas/`