Functional Programming in Python

MARCH 15<sup>TH</sup>, 2013
Write a program that counts the number of numbers in the range 0 through 1000 that contain the digit 7.

- The program in its entirety:

```python
def containsSeven(s):
    return "7" in s

print len(filter(containsSeven, map(str, range(0, 1001))))
```
Functional Programming

- *Functional programming* is a programming paradigm that treats computation as the evaluation of mathematical functions.
- Programming languages that do *not* use this style are called imperative programming languages (C, C++, Java, etc).
- For our purposes in this course, functional programming amounts to *passing functions as arguments to other functions*.
- We will learn about built-in Python functions *map*, *filter*, and *reduce* that are extremely powerful because they take other functions as arguments.
In general, it is easier to reason formally about programs written in functional programming style.

- General purpose functional programming languages: Lisp, Scheme, Haskell, OCaml, etc.
- Specialized functional programming languages: Mathematica (mathematical computation), R (statistical computation), etc.

Python has elements of both imperative style and functional style.
The `map` function

- `map(f, [a, b, c, d, e])` returns the list `[f(a), f(b), f(c), f(d), f(e)]`
- The first argument of `map` is a function `f` and the second argument is a list `L`; it returns a new list obtained by applying `f` onto every element of `L`.

**Examples:**
- `map(round, [4.57, -9.876, math.pi])` returns `[5.0, -10.0, 3.0]`
- `map(str, range(0, 6))` returns `['0', '1', '2', '3', '4', '5']`

- The `map` function allows us to construct new lists from old ones.
The `map` function

- Note that one can equivalently use the `for`-loop or the `while`-loop. Using the `map` function is faster.
- The `map` function can also take functions with more than one argument.

**Example:**

```python
def pow(x, y):
    return x + y

>>> map(pow, [3, 4, 5], [5, 6, 7])
[8, 10, 12]
```
The `filter` function

- `filter(f, L)` returns a sublist of `L` consisting of those elements in `L` (in the same order as they appear in `L`) for which the boolean function `f` evaluates to `True`.

**Examples:**
- `filter(bool, [0, -10, 0.0, None, "hello"])` returns `[-10, 'hello']`

- `filter(containsSeven, map(str, range(1001)))` returns a list containing all of the numbers in the range 0 through 1000 that contain 7.
The **reduce** function

- This function is used as:
  \[ \text{reduce}(f, L) \]

- Here \( f \) is a two-argument function and \( L \) is a list.
- At each step, **reduce** passes the current answer followed by the next item from the list, to \( f \) in order to obtain the next answer.
- By default, the first item in the sequence is taken as the first answer.

**Example:** \( \text{reduce}(\text{multiply}, \text{range}(1, n+1)) \) is a compact and efficient way of computing \( n! \).
Try these Examples!

- `map(str, range(0,10,3))`
- `len(filter(isPrime, range(20)))`
- `reduce(concat, map(str, range(1, 10, 2)))`
- `reduce(concat, range(1, 10, 2))`
- `map(range, range(5))`

- `isPrime` is a boolean function indicating primality.
- `concat(a, b)` returns `a + b`
The future of `map`, `filter`, and `reduce`

- There is some pushback against using elements of functional programming in Python.

- Python 3.0+ de-emphasizes these functions. In fact, `reduce` is not available as a built-in in Python 3.0+.

- Instead, users are encouraged to use `list comprehensions`, which are also available in Python 2.7.

- The next lecture is devoted to list comprehensions.