Iterators, Linked Lists, MapReduce, Dictionaries, and List Comprehensions... OH MY!

Special thanks to Scott Shawcroft, Ryan Tucker, and Paul Beck for their work on these slides.

Iterator

- Two special methods: `__iter__` and `__next__`
- `__iter__` sets up the iteration
- `__next__` returns values one by one until the end of the sequence is reached
- `raise StopIteration` to stop iteration
- now objects can be the target of for-each loops!

- may also define method: `x.__contains__(y)`
- supports sweet syntax sugar: `y in x`

Iterator Examples

```
>>> class IterList:
...     def __init__(self):
...         self.list = [2,4,6,8]
...
...     def __iter__(self):
...         self.iter_count = 0
...         return self
...
...     def __next__(self):
...         if (self.iter_count >= len(self.list)):
...             raise StopIteration
...         return self.list[self.iter_count]
>>> list = IterList()
>>> for element in list:
...     print(element*2)
[4, 8, 12, 16]
```

Map

```
map(function, iterable, ...)
```

- Map applies **function** to each element of **iterable** and creates a list of the results
- You can optionally provide more iterables as parameters to map and it will place tuples in the result list
- Map returns an iterator which can be cast to list

python™

Map Examples

```
>>> nums = [0, 4, 7, 2, 1, 0, 9, 3, 5, 6, 8, 0, 3]
>>> nums = list(map(lambda x : x % 5, nums))
>>> print(nums)
[0, 4, 2, 2, 1, 0, 4, 3, 0, 1, 3, 0, 3]
>>> def even (x):
...     if (x % 2 == 0):
...             return "even"
...     else:
...             return "odd"
>>> list (map(even, nums))
['even', 'even', 'odd', 'even', 'odd', 'even', 'odd',
'odd', 'odd', 'even', 'even', 'even', 'odd']
```

python

Functions as Parameters

Functions can be assigned to variables and/or passed as parameters

```
>>> list = ['once', 'upon', 'a', 'time', 'in', 'a']
>>> def foo (x):
...     return x * 3
>>> bar = foo
>>> my_map (foo, list)
['onceonceonce', 'uponuponupon', 'aaa', 'timetimetime',
'ininin', 'aaa']
>>> my_map (bar, list)
['onceonceonce', 'uponuponupon', 'aaa', 'timetimetime',
'ininin', 'aaa']
```

Map Code

```
>>> def my_map (fun, list):
...     nlist = []
...     for item in list:
...             nlist.append(fun(item))
...     return nlist
```

Reduce

`reduce(`**`function, iterable`**`[,`**`initializer`**`])`

- Reduce will apply **function** to each element in **iterable** along with the sum so far and create a cumulative sum of the results
- **function** must take two parameters
- If initializer is provided, initializer will stand as the first argument in the sum
- Unfortunately in python 3 reduce() requires an import statement
  - `from functools import reduce`

Reduce Examples

```
>>> nums = [9, 2, 0, -4, 0, 0, 7, 5, 3, 8]
>>> reduce(lambda sum, current: sum + current, nums)
30
>>> foo = ['in', 'a', 'galaxy', 'far', 'far', 'away']
>>> reduce(lambda sum, current : sum + current, foo)
'inagalaxyfarfaraway'
>>> reduce(lambda x,y : x+y + “ “, foo, “once upon a time ”)
'once upon a time in a galaxy far far away '
```

Reduce Examples

```
>>> numlists = [[1, 2, 3], [4, 5], [6, 7, 8, 9]]
>>> reduce(lambda sum, current: sum+current, numlists, [])
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> nums = [1, 2, 3, 4, 5, 6, 7, 8]
>>> nums = list(reduce(lambda x, y : (x, y), nums))
>>> print(nums)
(((((((1, 2), 3), 4), 5), 6), 7), 8)
```

python™

Reduce Problem

Goal: given a list of numbers I want to find the average of those numbers in a few lines using **`reduce`**

For Loop Method:

- sum up every element of the list
- divide the sum by the length of the list

Reduce Problem

Solution

```
>>> nums = [92, 27, 63, 43, 88, 8, 38, 91, 47, 74,
...             18, 16, 29, 21, 60, 27, 62, 59, 86, 56]
average = reduce(lambda x, y : x + y, nums) / len(nums)
```

python™

MapReduce

Framework for processing huge datasets on certain kinds of distributable problems

Map Step:

- master node takes the input, chops it up into smaller and distributes to smaller nodes

- smaller nodes may continue chopping recursively

Reduce Step:

- master node then takes the answers to all the sub-problems and combines them in a way to get the desired output

MapReduce

Problem: Given an email how do you tell if it is spam?

Count occurrences of certain words. If they occur too frequently the email is spam.

MapReduce

```
>>> email = “Well, there's egg and bacon; egg sausage
and bacon; egg and spam; egg bacon and spam; egg bacon
sausage and spam; spam bacon sausage and spam; spam egg
spam spam bacon and spam; spam sausage spam spam bacon
spam tomato and spam;”
>>> email = email.split()
>>> def spamWeight (word):
        if ("spam" in word):
                return 1
        else:
                return 0
>>> list(map (spamWeight, email))
[0, 0, ... 1, 0, 0, 1]
>>> reduce(lambda x, xs: x + xs, map(spamWeight, email))
14
```

python™

Dictionaries

map keys, which can be any immutable type, to values, which can be any type

```
>>> # declaring an empty dictionary
>>> eng2sp = {}
>>> # adding values
>>> eng2sp['one'] = 'uno'
>>> eng2sp['two'] = 'dos'
>>> # declaring dictionary with initial values
>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three':'tres'}
```

python™

Dictionaries

Operations:

- print, del, len, in

Methods:

- keys(), values(), items()

```
>>> eng2sp.items()
[('three', 'tres'), ('two', 'dos'), ('one', 'uno')]
```

List Comprehensions

```
[expression for element in list]
```

- Applies the expression to each element in the list
- You can have 0 or more for or if statements
- If the expression evaluates to a tuple it must be in parenthesis

List Comprehensions

```
>>> vec = [2, 4, 6]
>>> [3*x for x in vec]
[6, 12, 18]

>>> [3*x for x in vec if x > 3]
[12, 18]

>>> [3*x for x in vec if x < 2]
[]
>>> [[x,x**2] for x in vec]
[[2, 4], [4, 16], [6, 36]]

>>> [x, x**2 for x in vec]
SyntaxError: invalid syntax (requires parens)
```

python™

List Comprehensions

You can do most things that you can do with map, filter and reduce more nicely with list comprehensions

The email spam program from earlier using list comprehensions:

```
>>> email = ['once', 'upon', 'a', 'time', 'in', 'a', 'far']
>>> len( [1 for x in email if x == 'a'] )
2
```