Lecture 09
Filter, Map, Reduce, and Lambda

Barron has successfully reduced scratching, but the overhead is huge!
Recursion takeaways

- Any recursive algorithm can be implemented with iteration
- Recursion is a trade-off in efficiency vs. readability
- Avoid multiple recursive calls whenever possible
  - e.g., $O(n)$ vs. $O(2^n)$

Multiple base cases

- Not always an empty or singular sequence
  - e.g., Palindrome checker: front and back must be equal

Recursion vs. Iteration

- Is the Fibonacci sequence a good function to recurse in practice?
- Searching through directed graphs or file structures are better suited for recursion
Lecture 09 Goals

Lecture 09A:
1. Introduce *high-level* functions: `filter()`, `map()`, & `reduce()`
2. Introduce *anonymous* functions: `lambda`

Lecture 09B:
1. Introduction to Object Oriented Programming (OOP)
2. How to find help on objects
filter()

• A higher-order function

• Syntax:

```python
filter(function, sequence)
```

• applies `function` to each element of `sequence` and returns elements for which the function returns `true`

• filter returns a subset of `sequence`
  • to `generate` the actual list, we need to apply `list()`
filter() Examples

def isDivBy3(x): # is divisible by 3?
    return x % 3 == 0

def isEven(x): # is even?
    return x % 2 == 0

def isCap(s): # is first character capitalized?
    return 'A' <= s[0] <= 'Z'

>>> list(filter(isDivBy3, range(0,31)))
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30]

>>> list(filter(isEven,  filter(isDivBy3, range(0,31))))
[0, 6, 12, 18, 24, 30]

>>> list(filter(isCap, ['he','Martha','tree','George','chop']))
['Martha', 'George']

>>> list(filter(isCap, 'Martha Dandridge-Washington'))
["Martha Dandridge-Washington"]
filter() Examples

def isDivBy3(x): # is divisible by 3?
    return x % 3 == 0

def isEven(x): # is even?
    return x % 2 == 0

def isCap(s): # is first character capitalized?
    return 'A' <= s[0] <= 'Z'

>>> list(filter(isDivBy3, range(0,31)))
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30]

>>> list(filter(isEven, filter(isDivBy3, range(0,31))))
[0, 6, 12, 18, 24, 30]

>>> list(filter(isCap, ['he','Martha','tree','George','chop']))
['Martha', 'George']

>>> list(filter(isCap, 'Martha Dandridge-Washington'))
['M', 'D', 'W']
map() 

- A higher-order function

Syntax:

```python
map(function, sequence)
```

- applies `function` to each element of `sequence` and returns the `results`

As with `range`:

- you can think of `map` as producing a list
- in many cases it can be used like one
- to `generate` the actual list, we need to use `map()` with `list()`
map() Examples

```python
def triple(x):
    return 3*x

def square(x):
    return x*x

def first_char(s):
    return s[0]

>>> list(map(triple, [0, 1, 2, 3, 4, 5]))
[0, 3, 6, 9, 12, 15]

>>> list(map(square, range(6)))
[0, 1, 4, 9, 16, 25]

>>> list(map(first_char, ['python', 'is', 'fun!']))
[???, ???, ???”

>>> list(map(triple, 'python'))
[???, ???, ???”
```
map() Examples

```python
def triple(x):
    return 3*x

def square(x):
    return x*x

def first_char(s):
    return s[0]

>>> list(map(triple, [0, 1, 2, 3, 4, 5]))
[0, 3, 6, 9, 12, 15]

>>> list(map(square, range(6)))
[0, 1, 4, 9, 16, 25]

>>> list(map(first_char, ['python', 'is', 'fun!']))
['p', 'i', 'f']

>>> list(map(triple, 'python'))
['ppp', 'yyy', 'ttt', 'hhh', 'ooo', 'nnn']
```
reduce()

• Required: from functools import reduce

• Syntax:

```python
reduce(f, s)
```

• reduce continually applies the function $f(x, y)$ to the sequence $s$. It returns a single value.

For $s = [s_1, s_2, s_3, \ldots, s_n]$, $f(x, y)$ is applied to the first two elements. Note: $f()$ has 2 input parameters!

The list on which reduce() works looks now like this:

• $[f(s_1, s_2), s_3, \ldots, s_n]$, In the next step the list is

• $[f(f(s_1, s_2), s_3), \ldots, s_n]$

Continue like this until just one element is left and return this element as the result of reduce()
reduce() Examples

from functools import reduce

def add(x, y):
    return x+y

>>> reduce(add, [47, 11, 42, 13])
113

Calculated via add(add(add(47,11), 42), 13)
reduce() Examples

from functools import reduce

def add(x, y):
    return x+y

def mult(x, y):
    return x*y

>>> reduce(add, range(1,6))
15
>>> reduce(mult, range(1,6))
120
>>> reduce(add, ['Just', 'ice,', ' Now!'])
???
reduce() Examples

from functools import reduce

def add(x, y):
    return x+y

def mult(x, y):
    return x*y

>>> reduce(add, range(1,6))
15
>>> reduce(mult, range(1,6))
120
>>> reduce(add, ['Just', 'ice,', ' Now!'])
'Justice, Now!'
What will this code output?

```python
from functools import reduce

def mult(x, y):
    return x*y

def mystery(n):
    return reduce(mult, range(1,n+1))

print(mystery(4))
```

A. 4  
B. 12  
C. 24  
D. [4 12 24]  
E. none of the above
What will this code output?

```python
from functools import reduce

def mult(x, y):
    return x*y

def mystery(n):
    return reduce(mult, range(1,n+1))

print(mystery(4))
```

A. 4  
B. 12 
C. 24 
D. [4 12 24] 
E. none of the above
from math import log

def isOdd(x):
    return x%2 == 1

def odd_log_sum(n):
    return reduce(add, map(log, filter(isOdd,range(1, n+1))))

print(odd_log_sum(5))
Other Useful Built-In Functions

- **sum(list)**: computes & returns the sum of a list of numbers
  
  >>> sum([4, 10, 2])
  16

- Here's how we could define it recursively:

  ```python
def sum(values):
    ''' computes the sum of a list of numbers.
    input values: an arbitrary list of 0 or more #s
    '''
    if values == []:  # base case
      return 0
    else:
      sum_rest = sum(values[1:])  # recursive case
      return values[0] + sum_rest
  ```
Other Useful Built-In Functions

• `sum(list)`: computes & returns the sum of a list of numbers
  >>> sum([4, 10, 2])
  16

• Here's how we could define it using `reduce`:

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

def sum(vals):
    return reduce(add, vals)
```
Lambda Expressions and Anonymous Functions
Lambda Expressions

Python allows one to define functions in a single expression, i.e.,

```python
>>> isDivBy3 = (lambda x: x%3==1)
>>> list(filter(isDivBy3, range(0,31)))
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
```

Here keyword `lambda` indicates we’re defining a `function`, `x` is its `argument`, and `x%3==1` indicates the `return value`
Lambda Expressions

Python allows one to define functions in a single expression, i.e.,

```python
>>> isDivBy3 = (lambda x: x%3==1)
>>> list(filter(isDivBy3, range(0,31)))
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
```

Here keyword `lambda` indicates we’re defining a function, `x` is its argument, and `x%3==1` indicates the return value.

The code above is entirely equivalent to

```python
def isDivBy3(x): # is divisible by 3
    return x % 3 == 0

>>> list(filter(isDivBy3, range(0,31)))
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
```
Anonymous Functions

Python allows one to define functions in a single expression, i.e.,

```python
>>> isDivBy3 = (lambda x: x%3==1)
>>> list(filter(isDivBy3, range(0,31)))
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
```

Here we have assigned the definition of this function to the variable `isDivBy3`, but we could just as well have used it immediately

```python
>>> list(filter(lambda x: x%3==1, range(0,31)))
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
```

This last example is an example of the use of an **anonymous function** – it never had a name, but it did do its job.
Lambda practice

```python
def isEven(x): # is even?
    return x % 2 == 0

def isCap(s): # is first character capitalized?
    return 'A' <= s[0] <= 'Z'

>>> x = list(filter(lambda ________________, range(0,10)))
[0, 2, 4, 6, 8, 10]

>>> list(filter(lambda ________________, 
               ['he','Martha','tree','George','chop']))
['Martha', 'George']
```

23
Lambda practice

def isEven(x): # is even?
    return x % 2 == 0

def isCap(s): # is first character capitalized?
    return 'A' <= s[0] <= 'Z'

>>> x = list(filter(lambda x: x % 2 == 0, range(0,10)))
[0, 2, 4, 6, 8, 10]

>>> list(filter(lambda s: 'A' <= s[0] <= 'Z', ['he','Martha','tree','George','chop']))
['Martha', 'George']
Lambda practice

```python
>>> list(map(lambda _____________, [0, 1, 2, 3, 4, 5]))
[0, 3, 6, 9, 12, 15]

>>> list(map(lambda _____________, range(6)))
[0, 1, 4, 9, 16, 25]
```
Lambda practice

```python
>>> list(map(lambda x: 3*x, [0, 1, 2, 3, 4, 5]))
[0, 3, 6, 9, 12, 15]

>>> list(map(lambda x: x**2, range(6)))
[0, 1, 4, 9, 16, 25]
```
Lambda practice

```python
>>> list(map(lambda c: c[1], ['python', 'is', 'fun!']))
???

>>> list(map(lambda c: c*2, 'python'))
???
```
Lambda practice

```python
>>> list(map(lambda c: c[1], ['python', 'is', 'fun!']))
['y', 's', 'u']

>>> list(map(lambda c: c*2, 'python'))
['pp', 'yy', 'tt', 'hh', 'oo', 'nn']
```
>>> reduce(lambda ____________, range(1,6))
15
>>> reduce(lambda ____________, range(1,6))
120
>>> reduce(lambda ____________, \
    ['Just', 'ice,', 'Now!'])
'Justice, Now!'
Lambda practice

```python
>>> reduce(lambda x,y: x+y, range(1,6))
15
>>> reduce(lambda x,y: x*y, range(1,6))
120
>>> reduce(lambda x,y: x+y, ['Just', 'ice,',' Now!'])
'Justice, Now!'
```
When *not* to use anonymous functions

Anonymous functions
1. do not allow testing
2. do not support doc strings
3. can make code really, really confusing

Do not use complex anonymous functions, i.e. ones that are not readily understandable, or easily verifiable by inspection

Concise code is good

Opaque code is bad

When to use anonymous functions

1. There are no existing functions that do what you need

2. The function is trivial: the function doesn’t need a name

3. Having a lambda expression makes your code more understandable than the function names you can think of
Lambda Expression Summary

This function returns the sum of its two arguments

$$(\text{lambda } x, y: x+y)$$

Lambda functions can be used wherever function objects are required. They are syntactically restricted to a single expression.

Semantically, they are just syntactic sugar for a normal function definition, i.e., both definitions below are functionally the same

```python
add = (lambda x, y: x+y)

def add(x, y): # add two numbers
    return x+y
```
Use Filter, Map and/or Reduce to compute with a lambda function.

```python
def num_vowels(s):
    '''Returns the number of vowels in a string of letters'''
    #Hint: The function string.count(substring) returns the total number of times each substring appears in string

def mymax(values):
    '''returns the largest element in a non-empty list'''
Use Filter, Map and/or Reduce to compute with a lambda function

```python
def num_vowels(s):
    '''Returns the number of vowels in a string of letters'''
    return reduce(lambda x,y: x+y, map(s.lower().count, 'aeiou'))
```

```python
def mymax(values):
    '''returns the largest element in a non-empty list'''
    return reduce(lambda x,y: x if x > y else y, values)
```
Lecture 09B: Object Oriented Programming
Recall: Strings Are Objects

- In Python, a string is an object.
  - **attributes:**
    - the characters in the string
    - the length of the string
  - **methods:** functions inside the string that we can use to operate on the string

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>String object for 'hello'</th>
<th>String object for 'bye'</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper()</td>
<td>converts to uppercase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>replace()</td>
<td>replaces characters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lower()</td>
<td>converts to lowercase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>split()</td>
<td>splits string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>find()</td>
<td>finds substring</td>
<td></td>
<td></td>
</tr>
<tr>
<td>count()</td>
<td>counts occurrences</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Recall: String Methods (partial list)

- `s.lower()`: return a copy of `s` with all lowercase characters
- `s.upper()`: return a copy of `s` with all uppercase characters
- `s.find(sub)`: return the index of the first occurrence of the substring `sub` in the string `s` (-1 if not found)
- `s.count(sub)`: return the number of occurrences of the substring `sub` in the string `s` (0 if not found)
- `s.replace(target, repl)`: return a new string in which all occurrences of `target` in `s` are replaced with `repl`
Examples of Using String Methods

```python
>>> chant = 'We are the Bears!

>>> chant.upper()

>>> chant.lower()

>>> chant.replace('e', 'o')

>>> chant
```
Examples of Using String Methods

```python
>>> chant = 'We are the Bears!

>>> chant.upper()
'WE ARE THE BEARS!'

>>> chant.lower()
'we are the bears!

>>> chant.replace('e', 'o')
'Wo aro tho Boars!'  

>>> chant
'We are the Bears!'```
Splitting a String

- The `split()` method breaks a string into a list of substrings.

  ```python
  >>> name = 'Martin Luther King'
  >>> name.split()
  ???
  >>> components = name.split()
  >>> components[0]
  ???
  ```

- By default, it uses *whitespace characters* (spaces, tabs, and newlines) to determine where the splits should occur.

- You can specify a different separator:

  ```python
  >>> date = '11/10/2014'
  >>> date.split('/')
  ???
  ```
Splitting a String

- The `split()` method breaks a string into a list of substrings.

  ```python
  >>> name = 'Martin Luther King'
  >>> name.split()
  ['Martin', 'Luther', 'King']
  >>> components = name.split()
  >>> components[0]
  'Martin'
  
  >>> date = '11/10/2014'
  >>> date.split('/')
  ['11', '10', '2014']
  ```

- By default, it uses *whitespace characters* (spaces, tabs, and newlines) to determine where the splits should occur.

- You can specify a different separator:

  ```python
  >>> date = '11/10/2014'
  >>> date.split('/ ')
  ['11', '10', '2014']
  ```
Discovering What An Object Can Do

• Use the documentation for the **Python Standard Library**: [docs.python.org/3/library](https://docs.python.org/3/library)
Here's the section on the str type (the type of string objects):

```
class str(object=\"")
class str(object=b", encoding='utf-8', errors='strict')
```

Return a string version of object. If object is not provided, returns the empty string. Otherwise, the behavior of str() depends on whether encoding or errors is given, as follows.

If neither encoding nor errors is given, str(object) returns object.__str__(), which is the “informal” or nicely printable string representation of object. For string objects, this is the string itself. If object does not have a __str__() method, then str() falls back to returning repr(object).

If at least one of encoding or errors is given, object should be a bytes-like object (e.g.

```
str.capitalize()
```

Return a copy of the string with its first character capitalized and the rest lowercased.

```
str.casefold()
```

Return a casefolded copy of the string. Casefolded strings may be used for caseless matching.

Scrolling down shows us the available methods:
Discovering What An Object Can Do (cont.)

- Scrolling down, we can find info. about a method called `strip()`:

```python
str.strip([chars])
```

Return a copy of the string with the leading and trailing characters removed. The `chars` argument is a string specifying the set of characters to be removed. If omitted or `None`, the `chars` argument defaults to removing whitespace. The `chars` argument is not a prefix or suffix; rather, all combinations of its values are stripped:

```python
>>> 'spacious'.strip()
'spacious'
>>> 'www.example.com'.strip('cmowz.')
'exmaple'
```
What is the output of this program?

```python
s = '    programming   

s = s.strip()

s.upper()

s = s.split('r')

print(s)
```

A. `['    p', 'og', 'amming   ']
B. `['p', 'og', 'amming']`
C. `['    P', 'OG', 'AMMING   ']
D. `['P', 'OG', 'AMMING'][]
E. none of the above
What is the output of this program?

```python
s = '    programming   

s = s.strip()       # s = 'programming'
s.upper()           # 'PROGRAMMING' (no change to s!)
s = s.split('r')    # s = ['p', 'og', 'amming']

print(s)
```

A. ['    p', 'og', 'amming   ']
B. ['p', 'og', 'amming']
C. ['    P', 'OG', 'AMMING   ']
D. ['P', 'OG', 'AMMING']
E. none of the above