UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Lists & Higher Order Functions

Learning Objectives

- Learn three new common Higher Order Functions:
  - map, filter, reduce
- These each apply a function to a sequence (list) of data
- They are "lazy" so we may need to call `list()`

- Map: Transform each item
  - Input: A function and a sequence
  - Output: A sequence of the same length. The items may be different.

MAP

```
list(map(function_to_apply, list_of_inputs))
```

Transform each of items by a function.

e.g. square()

Inputs (Domain):

- Function
- Sequence

Output (Range):

- A sequence

```
def map(function, sequence):
    return [ function(item) for item in sequence ]

list(map(square, range(10)))
```

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Lists & Higher Order Functions: Filter

Learning Objectives

- Learn three new common Higher Order Functions:
  - map, filter, reduce
- These each apply a function to a sequence (list) of data
- map/filter are "lazy" so we may need to call `list()`

- Filter: Keeps items matching a condition.
  - Input: A function and sequence
  - Output: A sequence, possibly with items removed. The items don't change.

FILTER

```
list(filter(function, list_of_inputs))
```

Keeps each of item where the function is true.

Inputs (Domain):

- Function
- Sequence

Output (Range):

- A sequence

```
def filter(function, sequence):
    return [ item for item in sequence if function(item) ]
```

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Lists & Higher Order Functions
Reduce

Learning Objectives

- Learn three new common Higher Order Functions:
  - map, filter, reduce
- These each apply a function to a sequence (list) of data

- Reduce: “Combines” items together, probably doesn’t return a list.
  - Input: A 2 item function and a sequence
  - A single value

REDUCE

`reduce(function, list_of_inputs)`

Successively **combine** items of our sequence

- function: add(), takes 2 inputs gives us 1 value.

Inputs (Domain):

- Function, with 2 inputs
- Sequence

Output (Range):

- An item, the type is the output of our function.

Note: We must import reduce from `functools`!

```python
def reduce(function, sequence):
    result = function(sequence[0], sequence[1])
    for index in range(2, len(sequence)):
        result = function(result, sequence[index])
    return result
```

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Lists & Higher Order Functions
Acronym

Today’s Task: Acronym

```
Input: "The University of California at Berkeley"

Output: "UCB"

def acronym(sentence):
       """YOUR CODE HERE"""
```

P.S. Pedantry alert: This is really an *initialism* but that's rather annoying to say and type. ☺ (However, the code we write is the same, the difference is in how you pronounce the result.) The more you know!

Three super important HOFS

* For the builtin filter/map, you need to then call list on it to get a list.
If we define our own, we do not need to call list

```
list(map(function_to_apply, list_of_inputs))
```

Applies function to each element of the list

```
list(filter(condition, list_of_inputs))
```

Returns a list of elements for which the condition is true

```
reduce(function, list_of_inputs)
```

Applies the function, combining items of the list into a "single" value.

Bonus / Review

- Left over slides we didn’t get to.

What does this do?

```
list(map(capitalize,
    ['michael', 'Alex', 'Srinath', 'julia']
))

Assume capitalize('michael') == 'Michael'
```

A) `['michael', 'Alex', 'Srinath', 'julia']`
B) `['Michael', 'Alex', 'Srinath', 'Julia']`
C) []
D) Error
E) I'm lost.

What does this do?

```
list(filter(return_false,
    range(100)
))

Assume return_false(42) == False
```

A) `range(0, 100) # A standard range object`
B) `[0, 1, 2, … 96, 97, 98, 99]`
C) [ ]
D) Error
E) I'm lost.

UCB CS88 Fa20 L6

Higher Order Functions

- **Functions that operate on functions**
- **A function**

```
def odd(x):
    return x%2==1

odd(3)
True
```

- **A function that takes a function arg**

Why is this not 'odd' ?

```
def filter(fun, s):
    return [x for x in s if fun(x)]

filter(odd, [0,1,2,3,4,5,6,7])
[1, 3, 5, 7]
```

Higher Order Functions (cont)

- **A function that returns (makes) a function**

```
def leq_maker(c):
    def leq(val):
        return val <= c
    return leq
```

```
>>> leq_maker(3)
<function leq_maker.<locals>.leq at 0x1019d8c80>

>>> leq_maker(3)(4)
False

>>> filter(leq_maker(3), [0,1,2,3,4,5,6,7])
[0, 1, 2, 3]
```