UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Lists & Higher Order Functions

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Functions That Make Functions

Learning Objectives

- Learn how to use and create higher order functions:
- Functions can be used as data
- Functions can accept a function as an argument
- **Functions can return a new function**

Review: What is a Higher Order Function?

- A function that takes in another function as an argument

OR

- **A function that returns a function as a result.**

Higher Order Functions

- **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

>>> [x for x in range(7) if leq_maker(3)(x)]
[0, 1, 2, 3]
```

Demo

- PythonTutor Link

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.

Functional List Operations

- Goal: Transform a list, and return a new result
- We'll use 3 functions that are hallmarks of functional programming
- Each of these takes in a function and a sequence

Function Name	Action	Input arguments	Input Fn. Returns	Output List
`map`	Transform every item	1 argument (each item)	"Anything", a new item	List of the same length, but possibly new values
`filter`	Return a list with fewer items	1 argument (each item)	A Boolean	List with possibly fewer items, but values are the same
`reduce`	"Combine" items together	2 arguments (current item, and the previous result)	Type should match the type each item	Usually a "single" item

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`!

```
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.

Functional List Operations

- Goal: Transform a list, and return a new result
- We'll use 3 functions that are hallmarks of functional programming
- Each of these takes in a function and a sequence

Function Name	Action	Input arguments	Input Fn. Returns	Output List
`map`	Transform every item	1 argument (each item)	"Anything", a new item	List of the same length, but possibly new values
`filter`	Return a list with fewer items	1 argument (each item)	A Boolean	List with possibly fewer items, but values are the same
`reduce`	"Combine" items together	2 arguments (current item, and the previous result)	Type should match the type each item	Usually a "single" item