



UC Berkeley EECS Lecturer Michael Ball

# 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

# 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.





#### 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

# 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 | Computer Science 88 | Michael Ball | http://cs88.org
```





UC Berkeley EECS Lecturer Michael Ball

# 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. <sup>(C)</sup> (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)
))
```

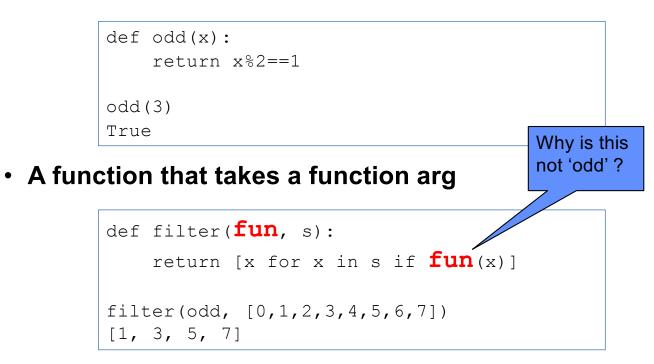
```
Assume return_false(42) == False
```

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



# **Higher Order Functions**

- Functions that operate on functions
- A function







# **Higher Order Functions (cont)**

• A function that returns (makes) a function

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

```
>>> 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]
```