



UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science



Week 3: HOFs



Announcements

- Watch Ed for announcements
 - One form all assignment extensions
 - *Please don't fill this out for slip days.*
 - If you need up to 3 days, just submit late. 😊
- CSM sections are out tomorrow
 - Totally optional, but lots of good prep.



UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science



List Comprehensions



Learning Objectives

- List comprehensions let us build lists “inline”.
- List comprehensions are an *expression that returns a list*.
- We can easily “filter” the list using a conditional expression, i.e. `if`



Data-driven iteration

- describe an expression to perform on each item in a sequence
- let the data dictate the control
- In some ways, nothing more than a concise for loop.

```
[ <expr with loop var> for <loop var> in <sequence expr > ]
```

```
[ <expr with loop var> for <loop var> in <sequence expr >  
if <conditional expression with loop var> ]
```

Demo!





UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science



Higher Order 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



Code is a Form of Data

- Numbers, Strings: All kinds of data
- Code is its own kind of data, too!
- Why?
 - More expressive programs, a new kind of abstraction.
 - ”Encapsulate” logic and data into neat packages.
- This will be one of the trickier concepts in CS88.



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.



Brief Aside: `import`

- Python organizes code in modules
 - These functions come with Python, but you need to "import" them.
 - `import module_name`
 - gives us access to `module_name` and `module_name.x`
 - `import module_name as my_module`
 - can access `my_module` and `my_module.x` (same code, just a different name)
 - `from module_name import x, y, z`
 - can only access the functions we import. `x` is `my_module.x`
- ```
from math import pi, sqrt
from operator import mul
```



## An Interesting Example

---

$$\sum_{k=1}^5 k = 1 + 2 + 3 + 4 + 5 = 15$$

$$\sum_{k=1}^5 k^3 = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 225$$

$$\sum_{k=1}^5 \frac{8}{(4k-3) \cdot (4k-1)} = \frac{8}{3} + \frac{8}{35} + \frac{8}{99} + \frac{8}{195} + \frac{8}{323} = 3.04$$



UC Berkeley EECS  
Lecturer  
Michael Ball

# Computational Structures in Data Science

---



## Higher Order 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

---





UC Berkeley EECS  
Lecturer  
Michael Ball

# Computational Structures in Data Science

---



## Environments & Higher Order 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**



## Example: compose

---

- Python Tutor:

```
http://pythontutor.com/composingprograms.html#code=def%20square%28x%29%3A%0A%20%20%20%20return%20x%20*%20x%0A%20%20%20%20%0As%20%3D%20square%0Ax%20%3D%20s%283%29%0A%0Adef%20make_adder%28n%29%3A%0A%20%20%20%20def%20adder%28k%29%3A%0A%20%20%20%20%20%20%20%20%20%
```



## Environment Diagrams

---

- Organizational tools that help you understand code
- **Terminology:**
  - **Frame:** keeps track of variable-to-value bindings, each function call has a frame
  - **Global Frame:** global for short, the starting frame of all python programs, doesn't correspond to a specific function
  - **Parent Frame:** The frame of where a function is defined (default parent frame is global)
  - **Frame number:** What we use to keep track of frames,  $f_1$ ,  $f_2$ ,  $f_3$ , etc
  - **Variable vs Value:**  $x = 1$ .  $x$  is the **variable**, 1 is the **value**



## Environment Diagrams Steps

---

1. Draw the global frame
2. When evaluating assignments (lines with single equal), always evaluate right side first
3. When you call a function MAKE A NEW FRAME!
4. When assigning a primitive expression (number, boolean, string) write the value in the box
5. When assigning anything else, draw an arrow to the value
6. When calling a function, name the frame with the intrinsic name – the name of the function that variable points to
7. The parent frame of a function is the frame in which it was defined in (default parent frame is global)
8. If the value isn't in the current frame, search in the parent frame



## Environment Diagram Tips / Links

---

- NEVER EVER draw an arrow from one variable to another.
- Useful Resources:
  - [http://markmiyashita.com/cs61a/environment\\_diagrams/rules\\_of\\_environment\\_diagrams/](http://markmiyashita.com/cs61a/environment_diagrams/rules_of_environment_diagrams/)
  - <http://albertwu.org/cs61a/notes/environments.html>