Lecture 5: Higher-Order Functions

Brian Hou
June 27, 2016

Announcements

• Homework 2 is due Wednesday 6/29
• Project 1 is due Thursday 6/30
• Earn 1 EC point for completing it by Wednesday 6/29
• Quiz 2 is on Thursday 6/30 at the beginning of lecture
• Environment Diagrams and Higher-Order Functions
• Group Tutoring is available! See Piazza for details

Roadmap

Introduction
Functions
Data
Mutability
Objects
Interpretation
Paradigms
Applications

Higher-Order Functions

Generalizing Computations (demo)

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

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

\[
\sum_{k=1}^{5} \frac{k(k+1)}{(4k-3)(4k+1)} \cdot 8 = \frac{8}{3} + \frac{8}{99} + \frac{8}{195} + \frac{8}{321} = 3.04
\]

Generalizing Computations (demo)

```python
def sum_naturals(n):
    total, k = 0, 1
    while k <= n:
        total, k = total + k, k + 1
    return total

def sum_cubes(n):
    total, k = 0, 1
    while k <= n:
        total, k = total + pow(k, 3), k + 1
    return total
```
Summation Example

```python
cube = lambda k: pow(k, 3)
def summation(n, term):
    """Sum the first N terms of a sequence."
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

>>> summation(5, cube)
225
```

Locally Defined Functions  (demo)

- Functions defined within other function bodies are bound to names in a local frame
  ```python
def make_adder(n):
    """Return a function that takes one argument k and returns k + n."
    def adder(k):
        return k + n
    return adder

>>> add_three = make_adder(3)
>>> add_three(4)
7
```

Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language

Higher-order function:
1. A function that takes a function as an argument value or
2. A function that returns a function as a return value

Higher-order functions:
- Express general methods of computation
- Remove repetition from programs
- Separate concerns among functions

Nested Definitions  (demo)

- Every user-defined function has a parent frame
- The parent of a function is the frame in which it was defined
- Every local frame has a parent frame
- The parent of a frame is the parent of the function called

Environments (Round 2)
Environment Diagram Rules (version 2)

Rules for `def` Statements:
1. Create a function with signature `<name>(<parameters>)` and parent `[parent=<label>]` (parent is the current frame)
2. Set the body of that function to be everything indented after the first line
3. Bind `<name>` to that function in the current frame

Rules for calling user-defined functions:
1. Create a new environment frame
2. Copy the parent of the function to the local frame: `[parent=<label>]`
3. Bind the function's parameters to its arguments in that frame
4. Execute the body of the function in the new environment

Environment Diagram

Application: Currying

- `add` is a two-argument function that returns the sum of the two arguments
- `make_adder` is a one-argument function that returns a one-argument function that returns the sum of the two arguments
- Currying allows us to represent functions with multiple variables as chains of functions with single variables
- It is named after mathematician and logician Haskell Brooks Curry (who rediscovered it after Moses Schönfinkel):
  \[
  \lambda x, y : x + y + 1(3, 4) \\
  \lambda x : \lambda y : x + y + 1(3)(4)
  \]