

Lambdas

pollev.com/cs61a

Announcements

Lambda Expressions

Example from Wednesday: summation

```
def cube(k):  
    return pow(k, 3)
```

Function of a single argument
(*not called "term"*)

```
def summation(n, term):  
    """Sum the first n terms of a sequence.
```

A formal parameter that will
be bound to a function

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

```
225
```

```
"""
```

```
total, k = 0, 1
```

```
while k <= n:
```

```
    total, k = total + term(k), k + 1
```

```
return total
```

The cube function is passed
as an argument value

1 + 8 + 27 + 64 + 125

The function bound to term
gets called here

What about the natural
numbers?

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

(Demo)

Lambda practice

```
def cube(k):  
    return pow(k, 3)  
  
def summation(n, term):  
    """Sum the first n terms of a sequence.  
  
    >>> summation(5, cube)  
    225  
    """  
    total, k = 0, 1  
    while k <= n:  
        total, k = total + term(k), k + 1  
    return total
```

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

```
sum = summation(5, lambda x: x)  
print(sum)
```

$$\sum_{k=1}^5 \left(\frac{1}{2}\right)^k = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32}$$

Write the call to summation for this series:

```
sum = summation(5, _____)  
print(sum)
```

Lambda Expressions

```
summation(5, lambda x: pow(1/2, x))
```

An expression:
evaluates to a
function

A function

with formal parameter x
that returns the value of $\text{pow}(1/2, x)$

Important: No "return" keyword!

Must be a single expression

Equivalent ways of writing this:

Lambda expressions can
be used in assignment
statements

```
term = lambda x: pow(1/2, x)
```

```
summation(5, term)
```

All lambda expressions can
be re-written using a def
(not vice versa)

```
def term(x):
```

```
    return pow(1/2, x)
```

```
summation(5, term)
```

Lambda Environments

$$\sum_{k=1}^5 \left(\frac{1}{2}\right)^k = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32}$$

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

(Demo)

pollev.com/cs61a

Function Currying

Convert a function that takes multiple arguments
into a chain of functions that each take a single argument

`pow(1/2, 5)`



`curry(pow)(1/2)(5)`

(Demo)

Zero-Argument Functions

(Demo: Dice)

Lab 02 Q2: Higher-Order Functions

```
>>> def cake():
...     print('beets')
...     def pie():
...         print('sweets')
...         return 'cake'
...     return pie
...
>>> chocolate = cake()
beets
>>> chocolate
<function cake.<locals>.pie at ...>
>>> chocolate()
sweets
'cake'
```

```
>>> more_chocolate, more_cake = chocolate(), cake
sweets
>>> more_chocolate
'cake'
```

```
>>> def snake(x, y):
...     if cake == more_cake:
...         return chocolate
...     else:
...         return x + y
...
>>> snake(10, 20)
<function cake.<locals>.pie at ...>
>>> snake(10, 20)()
sweets
'cake'
>>> cake = 'cake'
>>> snake(10, 20)
30
```

Lambda Expressions Practice

Lambda and Def

Any program containing lambda expressions can be rewritten using def statements.

```

                                twice                square
>>> (lambda f: lambda x: f(f(x)))(lambda y: y * y)(3)
81

>>> def twice(f):
...     def g(x):
...         return f(f(x))
...     return g
...
>>> def square(y):
...     return y * y
...
>>> twice(square)(3)
81
```

Fall 2022 Midterm 1 Question 4(a)

(2.0 pt) Choose **all** correct implementations of `funsquare`, a function that takes a one-argument function `f`. It returns a one-argument function `f2` such that `f2(x)` has the same behavior as `f(f(x))` for all `x`.

```
>>> triple = lambda x: 3 * x
>>> funsquare(triple)(5)  # Equivalent to triple(triple(5))
45
```

A: `def funsquare(f):`
 `return f(f)`

D: `def funsquare(f):`
 `return lambda x: f(f(x))`

B: `def funsquare(f):`
 `return lambda: f(f)`

E: `def funsquare(f, x):`
 `return f(f(x))`

C: `def funsquare(f, x):`
 `def g(x):`
 `return f(f(x))`
 `return g`

F: `def funsquare(f):`
 `def g(x):`
 `return f(f(x))`
 `return g`