Lambdas

Announcements

Lambda Expressions

(Demo)

Example from Friday: summation

```
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
```

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

A formal parameter that will
be bound to a function

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
```

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

Write the call to summation for this series:

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

pollev.com/cs61a

Lambda Expressions

An expression: evaluates to a function

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

Important: No "return" keyword!

A function

with formal parameter x

that returns the value of pow(1/2, x)

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

https://pythontutor.com/cp/composingprograms.html#code=common_ratio%20%3D%201/2%0A%0Aterm%20%3D%20lambda%20x%3A%20pow%28common_ratio,%20x%29%0A%0Acommon_ratio%20%3D%201/3%0A%0Adef%20summation%28n,%20term%29%3A%0A%20%20%20%20%22%22%22Sum%20the%20first%20N%20terms%20of%20a%20sequence.
%0A%0A%20%20%20%20%3E%3E%3E%20summation%285,%20cube%29%0A%20%20%20%20225%0A%20%20%20%20%22%22%22%0A%20%20%20%20common_ratio%20%3D%201/4%0A%20%20%20%20total,%20k%20%3D%200,%201%0A%20%20%20%20while%20k%20%3C%3D%20n%3A%0A%20%20%20%20%20%20%20%20total,
%20k%20%3D%20total%20%2B%20term%28k%29,%20k%20%2B%201%0A%20%20%20%20return%20total%0A%0Acommon_ratio%20%3D%201/5%0A%0Aprint%28summation%285,%20term%29%29%0A&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

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 Functions

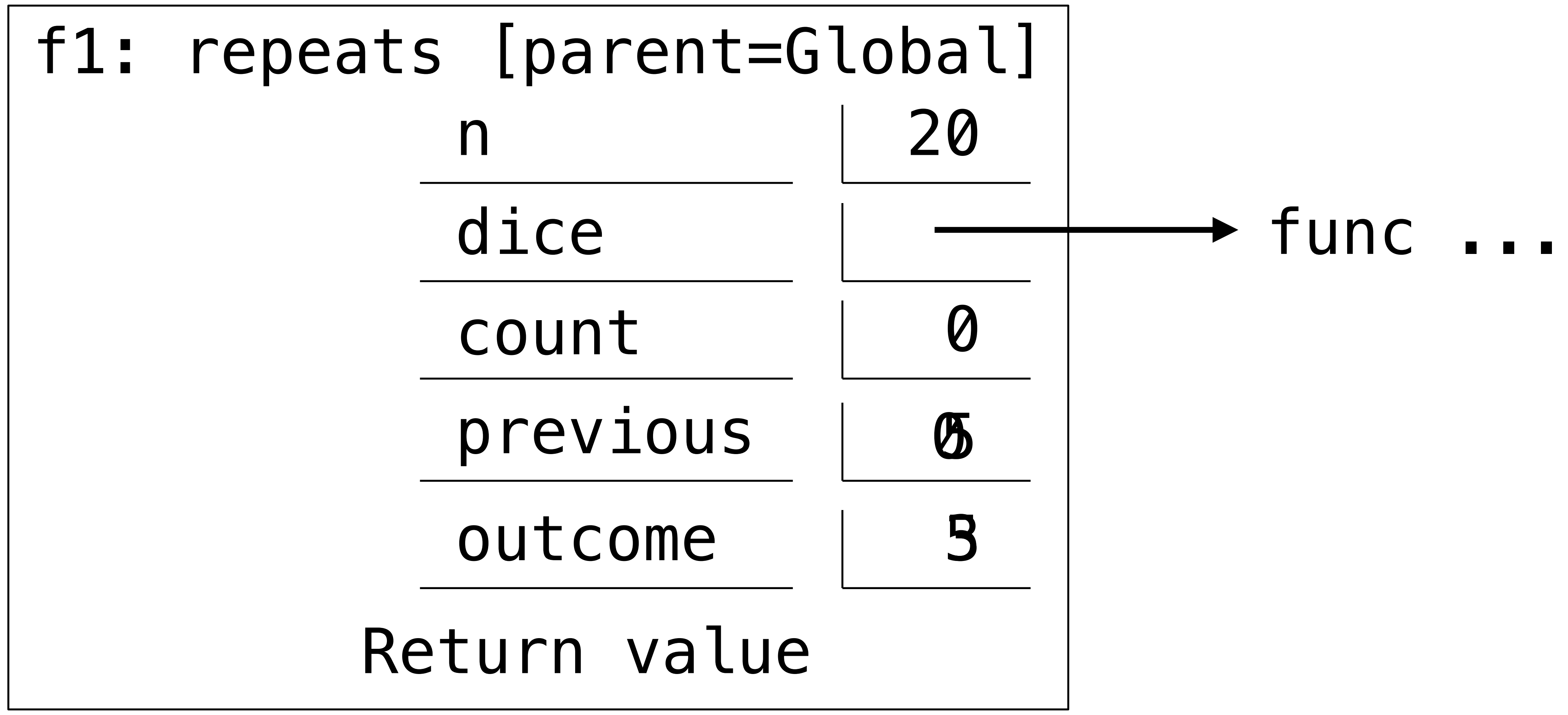
In the Hog project, there are multiple zero-argument functions that represent dice.

A dice function returns an integer that is the outcome of rolling once. (Demo)

Implement repeat, which returns the # of times in n rolls that an outcome repeats.

5 3 3 4 2 1 6 5 3 4 2 2 2 4 4 3 4 3 5 5   repeat(20, six_sided) -> 5

```
def repeats(n, dice):
    count = 0
    previous = 0
    while n:
        outcome = dice()
        if previous == outcome:
            count += 1
        previous = outcome
        n -= 1
    return count
```



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'
```

```
>>> 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
```

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

https://pythontutor.com/cp/composingprograms.html#code=def%20cake%28%29%3A%0A%20%20%20print%28'beets'%29%0A%20%20%20def%20pie%28%29%3A%0A%20%20%20%20%20%20%20print%28'sweets'%29%0A%20%20%20%20%20%20%20return%20'cake'%0A%20%20%20return%20pie%0A%0Achocolate%20%3D%20cake%28%29%0A%23%20chocolate%0A%23%20chocolate%28%29%0Amore_chocolate,
%20more_cake%20%3D%20chocolate%28%29,%20cake%0A%23%20more_chocolate%0Adef%20snake%28x,
%20y%29%3A%0A%20%20%20if%20cake%20%3D%3D%20more_cake%3A%0A%20%20%20%20%20%20%20return%20chocolate%0A%20%20%20else%3A%0A%20%20%20%20%20%20%20return%20x%20%2B%20y%0A%0A%23%20snake%2810,%2020%29%0Asnake%2810,%2020%29%28%29%0A%23%20cake%20%3D%20'cake'%0A%23%20snake%2810,%2020%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Lambda Expressions Practice

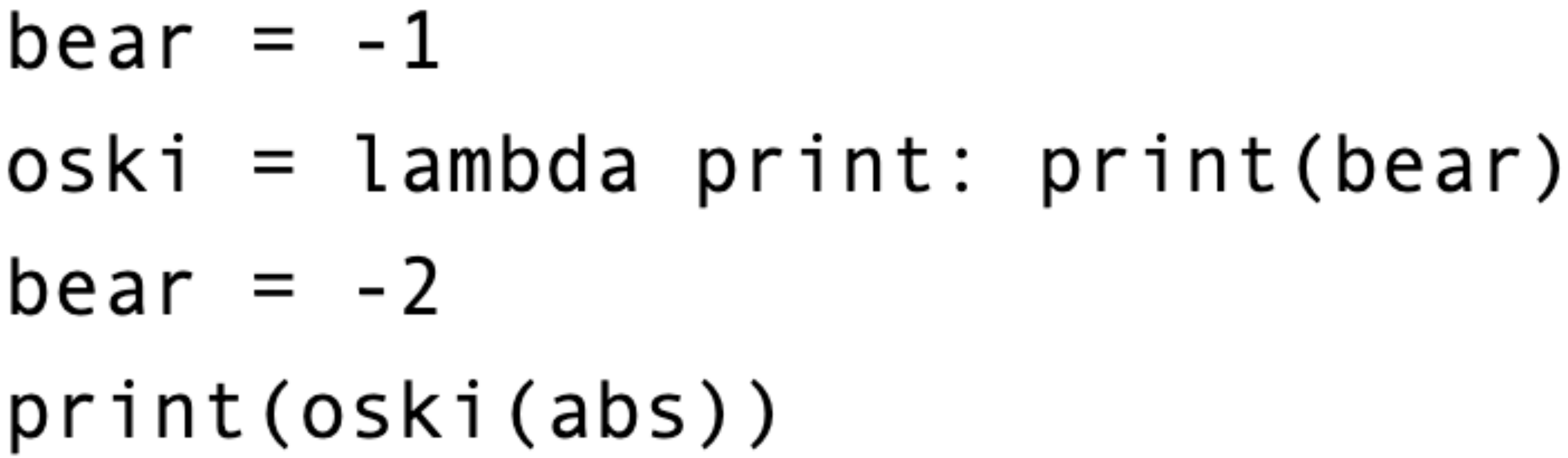
Lambda and Def

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 1(b)



```
bear = -1
oski = lambda print: print(bear)
bear = -2
print(oski(abs))
```

pollev.com/cs61a

https://pythontutor.com/cp/composingprograms.html#code=bear%20%3D%20-1%0Aoski%20%3D%20lambda%20print%3A%20print%28bear%29%0Abear%20%3D%20-2%0Aprint%28oski%28abs%29%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

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)
```

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

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

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

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

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

pollev.com/cs61a