Lambdas

Announcements

Lambda Expressions

Example from last week: 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
```

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

```
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)
```

pollev.com/c88c

Lambda Expressions

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

An expression: evaluates to a function

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/c88c

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%22%22%22Sum%20the%20first%20N%20terms%20of%20a%20sequence.%0A%0A%20%20%20%3E%3E%3E%20summation%285,%20cube%29%0A%20%20%20%20225%0A%20%20%20%22%22%22%0A%20%20%20common_ratio%20%3D%201/4%0A%20%20%20total,%20k%20%3D%200,%201%0A%20%20%20while%20k%20%3C%3D%20n%3A%0A%20%20%20%20%20%20total,%20k%20%3D%20total%20%2B%20term%28k%29,%20k%20%2B%201%0A%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

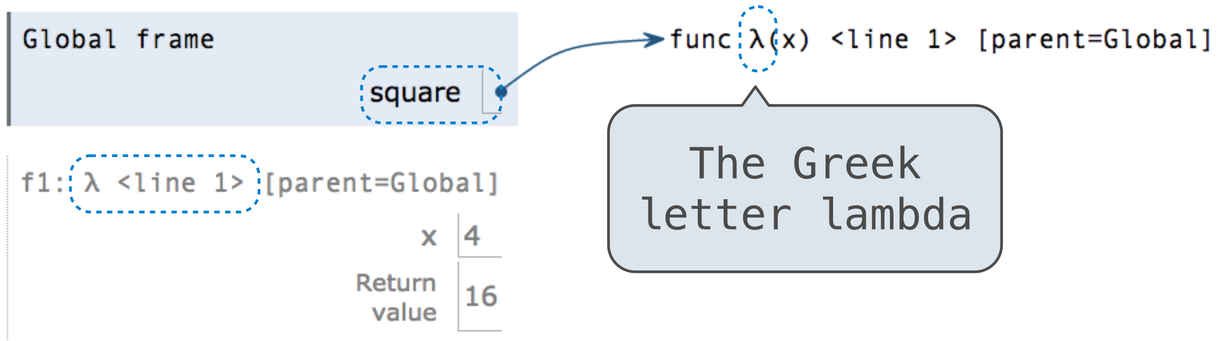
Lambda Expressions Versus Def Statements

```
square = lambda x: x * x
```

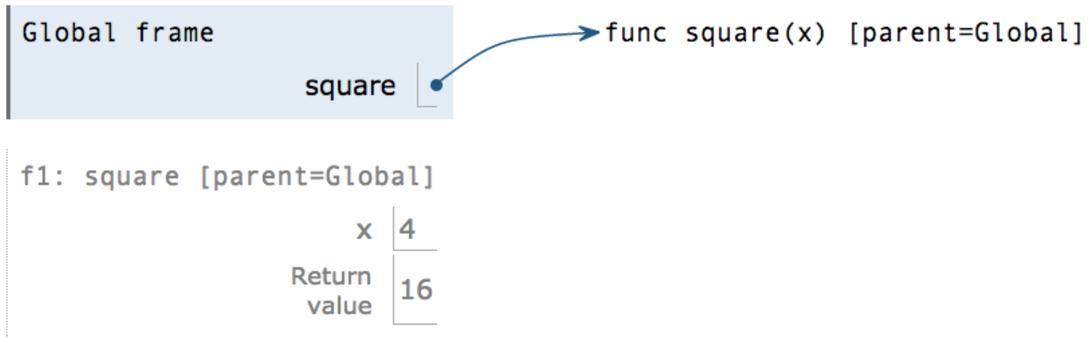
VS

```
def square(x):
    return x * x
```

- Both create a function with the same domain, range, and behavior.
- Both bind that function to the name square.
- Only the def statement gives the function an intrinsic name, which shows up in environment diagrams but doesn't affect execution (unless the function is printed).



(Demo)



Zero-Argument Functions

(Demo: Dice)

Lambda Expressions Practice

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