Sakire Arslan Ay

You can call me Shakira.

Cookie

Snow

Scholarly Associate Professor,

Coordinator for Software Engineering M.S. and B.S. programs

- Education:
  - Ph.D. in Computer Science – University of Southern California (2010)
  - M.S. in Computer Science – University of Southern California (2004)
  - B.S. in Computer Engineering – Bogazici University – Turkey (1999)
- Research Interests:
  - Large-Scale Geospatial Data Management and Indexing
  - Sensor-Rich Video Annotation and Search

- Courses at Washington State University
  - Software Engineering I
  - Programming Language Design
  - Introduction to Database Systems
  - Advanced Data Structures
  - Software Design Project I (Capstone)
  - Software Design Project II (Capstone)
  - Software Design

Higher Order Functions in Python

Sakire Arslan Ay

March 20, 2023

Review

We previously learned about:

- Functions in Python
- Basic data structures in Python: lists, tuples, and dictionaries
- Classes in Python
- Testing Python programs
  - Unit testing
  - Writing unit tests

Designing Functions

- Why do we include functions in our programs?

- What are the qualities of good functions?
  - Each function should have exactly one job.
  - Functions should be defined broadly.

```
sorted('Python')
sorted([2,8,1,-1,7,6])
sorted([('d',1),('e',4),('a',3)])
sorted([('d',1),('e',4),('a',3)], reverse = True)
sorted([('d',1),('e',4),('a',3)], reverse = False)
sorted([('d',1),('e',4),('a',3)], reverse = False, key=lambda t: t[1])
```

Source: https://composingprograms.com

DESIGNING FUNCTIONS

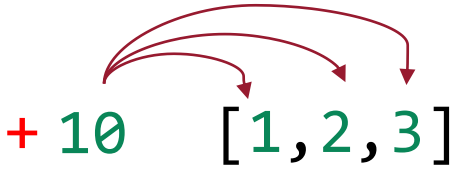
Generalization

- Generalizing functions with arguments
- Generalizing functions over computational processes

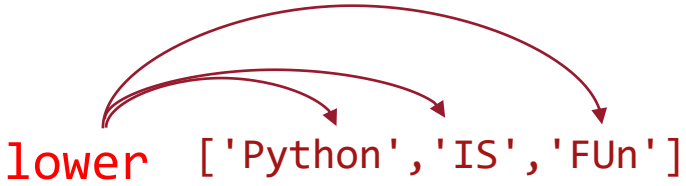
Generalizing over Computational Processes

- Common structure among problems – Example 1

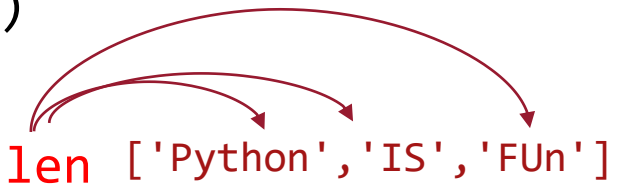
```
>>> add_to_all([1,2,3],10)
"""Adds 10 to each numeric value in
given list."""
```




```
>>> lower_all(['Python','IS','FUn'])
"""Converts each string in given list to
lowercase."""
```

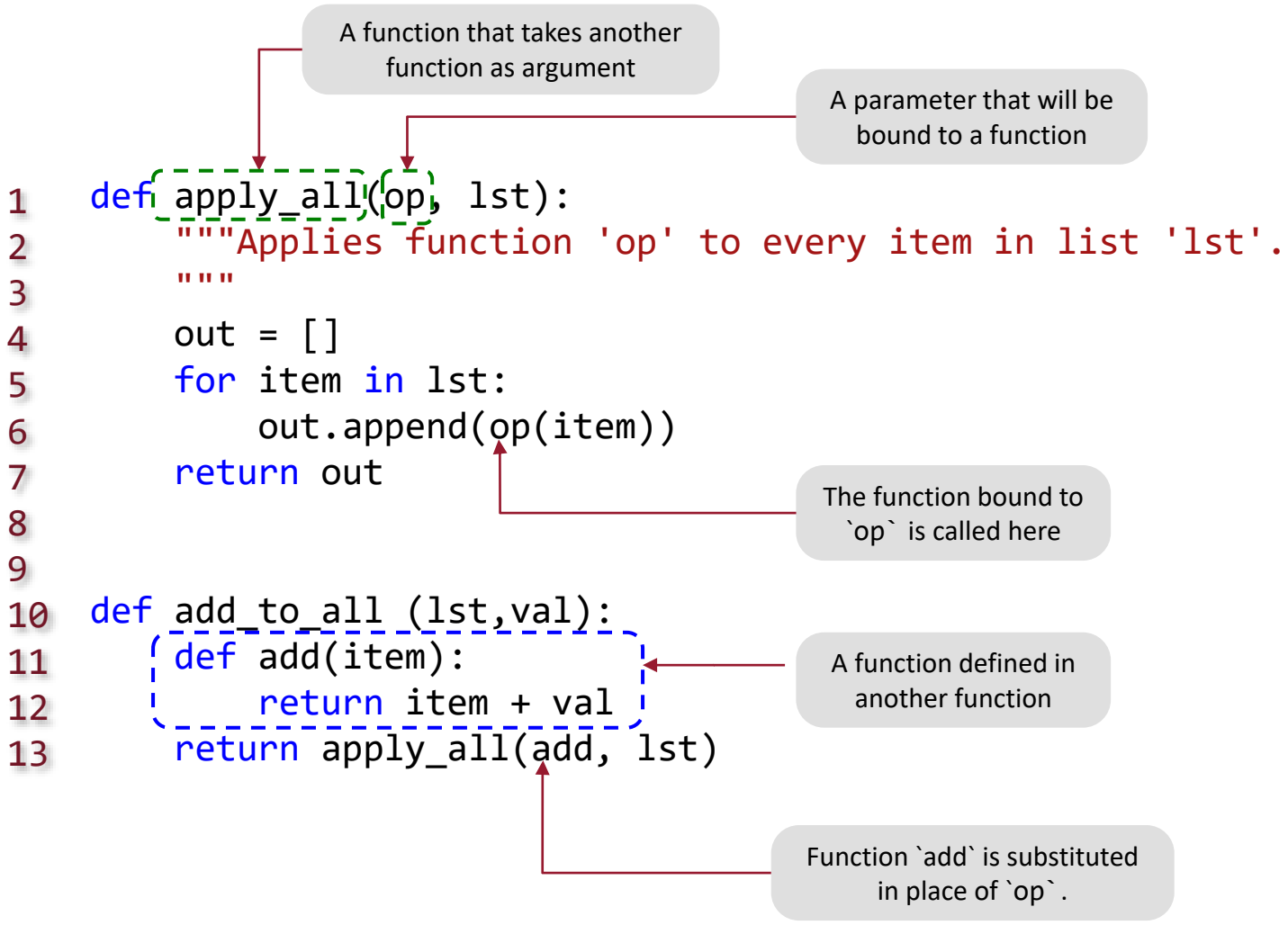



```
>>> length_of_all(['Python','IS','FUn'])
"""Calculates the length of each string in the
given list"""
```



— Demo!

Functions as Arguments



Python Functions

- In Python, functions that we define are just objects like anything else.

```
def meaningOfLife(x) :
    return "happiness" if x else "success"

type(meaningOfLife)
```

```
life = meaningOfLife

life(True)

d = {'mylife' : meaningOfLife}
```

Higher Order Functions

- In Python, functions are treated as *first-class values*.
  - Like any other value, functions can be passed as argument and returned as a result.
  - This provides a flexible way to compose programs.

- A function that **takes other functions as arguments** or **returns a function** is called a *higher order function*.

Built-in Higher Order Functions: map

- map takes a unary function *f* and an iterable input and applies *f* to every element in input. It returns result as a map object.
  - map(f,[x1,x2,…,xn]) ⇒ [f(x1),f(x2),…,f(xn)]

For example,

```
def pow2(x):
    return x**2

out = map(pow2, [1,2,3,4,5]) # returns a map object
list(out)
```

- Python's built-in map function is more general and faster.

```
def add(x,y):
    return x+y
list(map (add,[1,2,3,4],[5,6,7,8]))
```

returns ?

Checkpoint question?

- Are the following higher order functions?

(A)

```
def myFunction1(v,fn):

    def helper(x):
        return fn(v,x)

    return helper
```

(B)

```
def myFunction2(x,y):

    def helper(x):
        return x*x

    return helper(x)+helper(y)
```

(C)

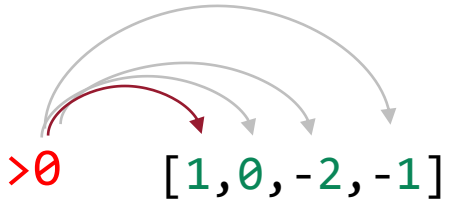
```
def myFunction3(f,y):
        return f(y)
```

Google forms link
https://tinyurl.com/gmulecture

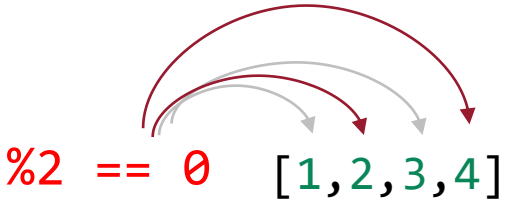
Generalizing over Computational Processes

- Common structure among problems – Example 2

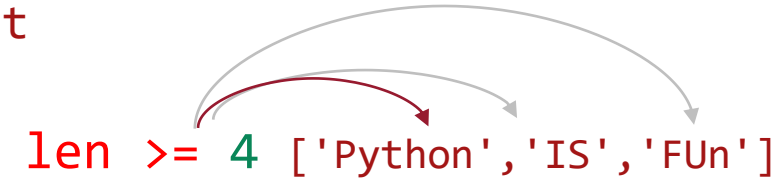
```
>>> get_positives([1,0,-2,-1])
"""Filters out the negative values
and zero from the given list."""
```




```
>>> get_evens([1,2,3,4,5,6])
"""Filters out the odd values from the
given list."""
```




```
>>> get_longer(['Python','IS','FUn'], 4)
"""Filters out the words in the list
shorter than the given length."""
```



— Demo!

Built-in Higher Order Functions: `filter`

filter

- `filter` takes a predicate function *p* (a unary function returning a `bool`) and an iterable `input`. It returns a filter object including the values from `input` for which *p* returns True.
  - `filter(p,[x1,x2,x3,…,xn])` ⇒ `[x2,…,xk]` (k<=n)

For example:

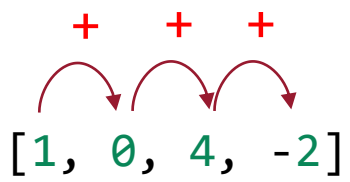
```
def is_positive(item):
    return item>0
list(filter(is_positive, [-4,3,1,-2,3,-5,1,9,0]))
```

returns ?

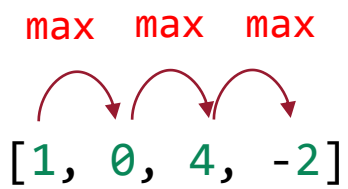
Generalizing over Computational Processes

- Common structure among problems – Example 3

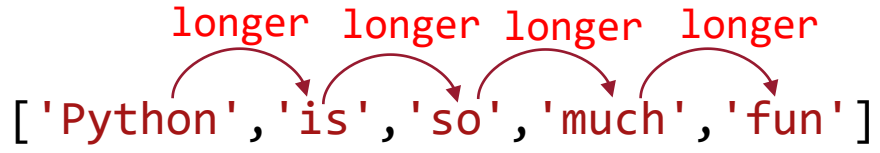
```
>>> sum_list([1, 0, 4, -2]
"""Sums all values in the given list."""
```




```
>>> max_of_list([1, 0, 4, -2]
"""Returns the maximum value from the
given list."""
```




```
>>> longest(['Python','is','so','much','fun'])
"""Returns the longest (leftmost) value
from the given list."""
```



— Demo!

Built-in Higher Order Functions: `reduce`

reduce

- `reduce` takes a binary function *f* and an iterable `input`. It combines the values in input by applying *f* and returns a single value.
  - `reduce(f,[x1,x2,x3,…,xn])` ⇒ `f( f(x1,x2),x3… ))`

For example:

```
from functools import reduce

reduce( max, [4,2,-3,8,6] )
```

- Unlike `map` and `filter` (which are defined and automatically imported from the builtins module) we must import reduce from the `functools` module explicitly.

Python Functions and Lambdas

- So far, each time we defined a new function, we gave it a name.
- Python provides a shortcut for quickly defining *anonymous* functions that define an expression and return it, called "*lambdas*":

```
def add(x,y):
    return (x+y)

lambda x , y  :   x + y
# A function that takes x and y and returns (x+y)
```

```
def meaningOfLife(x) :
    return "happiness" if x else "success"

meaningOfLife = lambda x: "happiness" if x else "success"
meaningOfLife(3)
```

Built-in Higher Order Functions: reduce

- How to use reduce?
  - Without the (optional) `init` argument:

```
reduce(lambda x,y : x+y , [1,2,3,4,5,6])  #returns 21

reduce(min, [4,2,3,8,6,1,1] )  #returns 1

reduce(lambda x,y: x[1]+y[1], [('Feb',10),('Mar',15),('Apr',20)]) #??
```

won't work

  - With the `init` argument:

```
reduce(lambda x,y : x+y, [1,2,3,4,5,6], 0)  #returns 21

reduce(min, [4,2,3,8,6,1,1], 0)  #returns 0

reduce(lambda x,y: x + y[1], [('Feb',10),('Mar',15),('Apr',20)], 0)
```

will return the sum of second values in tuples

Why are higher order functions useful?

- Enable code re-use.
- Help write precise, clean, and readable code.

FUNCTIONS AS RETURN VALUES

Functions as Return Values

- Consider the following function:

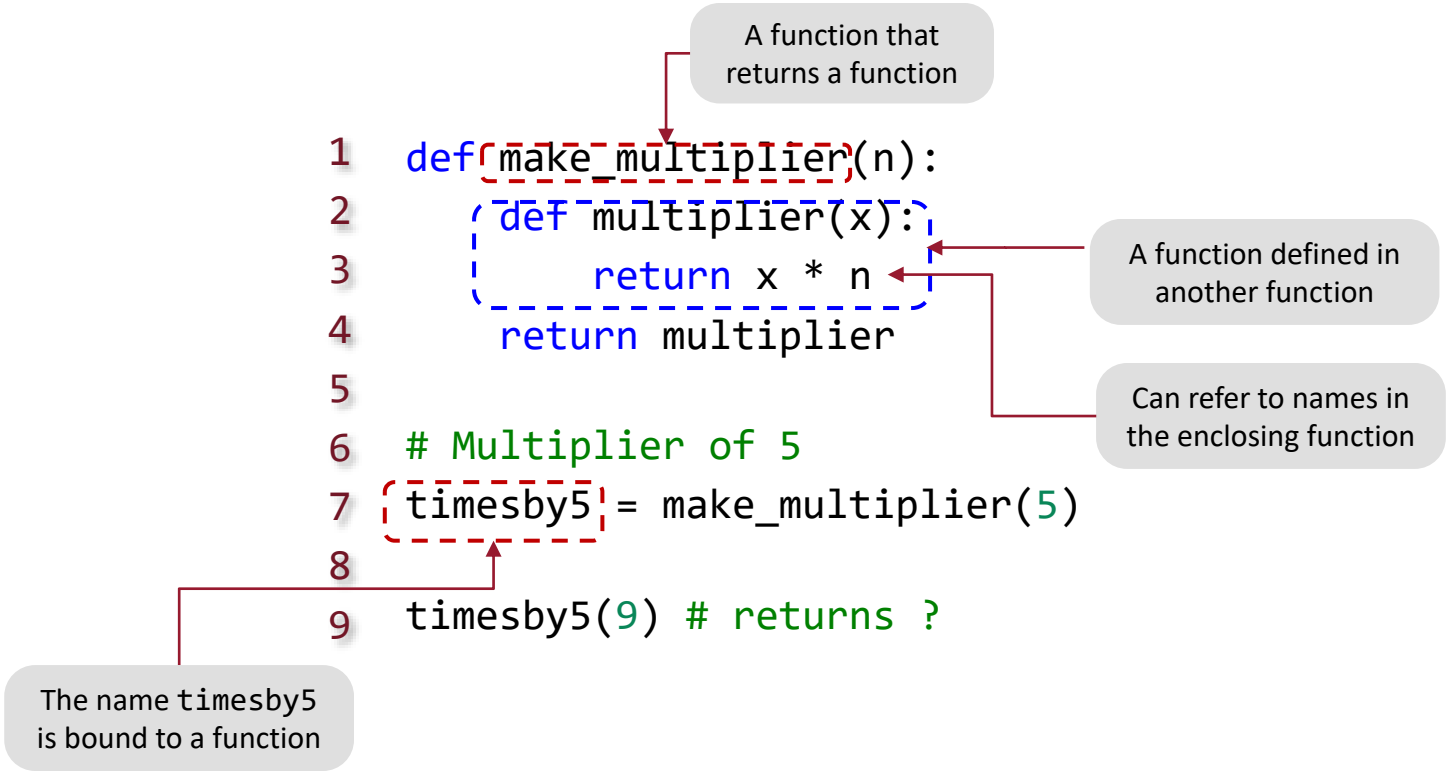
```
1   def make_multiplier(n):
2       def multiplier(x):
3           return x * n
4       return multiplier
5
6
7   # Multiplier of 5
8   timesby5 = make_multiplier(5)
9
10  # Multiplier of 10
11  timesby10 = make_multiplier(10)
12
13  timesby5(9) # returns ?
14
15  timesby10(3) # returns ?
```

1. Is `make_multiplier` a higher order function?
2. What will the following evaluate to ?

   `timesby5(9)`

   `timesby10(3)`

Functions as Return Values

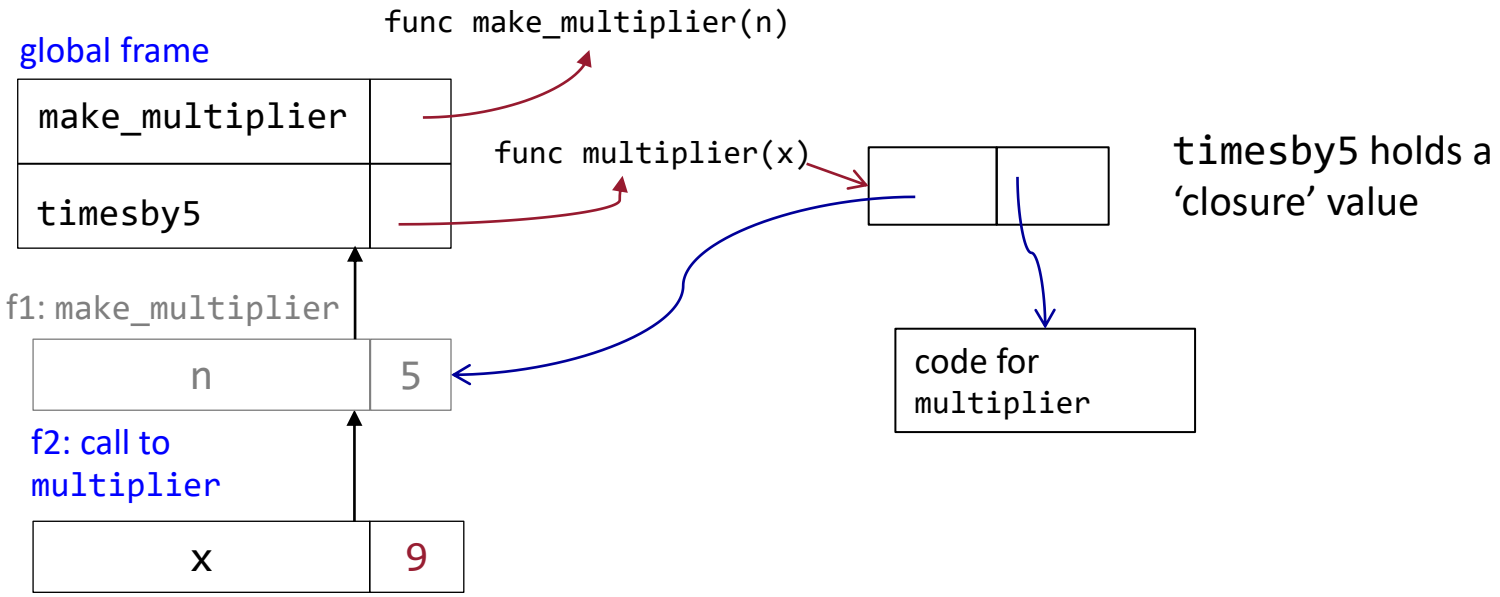


Python Tutor visualizer : https://tinyurl.com/pythontutorlink

Functions as Return Values

```
1  def make_multiplier(n):
2      def multiplier(x):
3          return x * n
4      return multiplier
5
6  # Multiplier of 5
7  timesby5 = make_multiplier(5)
8
9  timesby5(9) # returns ?
```

- A function value has two parts
  - the code (obviously)
  - environment that was current when the function was defined
- This pair is called a **function closure**



Questions?