

# Lab 4: Lists

Due at 11:59pm on 02/18/2015.

## Starter Files

---

Download [lab04.zip](#). Inside the archive, you will find starter files for the questions in this lab, along with a copy of the [OK](#) autograder.

## Submission

---

By the end of this lab, you should have submitted the lab with `python3 ok --submit`. You may submit more than once before the deadline; only the final submission will be graded.

- To receive credit for this lab, you must complete Questions 4, 5, 6, and 8 in [lab04.py](#) and submit through OK.
- Questions 9, 10, 11, 12, and 13 are *extra practice*. They can be found in the [lab04\\_extra.py](#) file. It is recommended that you complete these problems on your own time.

## Table of Contents

---

- [Lists](#)
  - [Question 1: What would Python print? List indexing](#)
  - [Question 2: What would Python print? List slicing](#)
  - [Question 3: What would Python print? List operations](#)
  - [Question 4: Fill in the blanks](#)
  - [Question 5: Reverse \(recursively\)](#)
  - [Question 6: Merge](#)
- [List Comprehensions](#)
  - [Question 7: What Would Python Print?](#)
  - [Question 8: Perfect squares](#)
- [Extra Questions](#)
  - [Question 9: Reverse \(iteratively\)](#)
  - [Question 10: Mergesort](#)
  - [Question 11: Coordinates](#)
  - [Question 12: Deck of cards](#)

- [Question 13: Adding matrices](#)

## Lists

---

### Question 1: What would Python print? List indexing

Predict what Python will display when you type the following into the interpreter. Then try it to check your answers.

```
>>> x = [1, 2, 3]
```

```
>>> x[0]
```

```
_____
```

```
>>> x[x[0]]
```

```
_____
```

```
>>> x[x[x[0]]]
```

```
_____
```

```
>>> x[3]
```

```
_____
```

```
>>> x[-1]
```

```
_____
```

```
>>> x[-3]
```

```
_____
```

[Toggle Solution](#)

### Question 2: What would Python print? List slicing

Predict what Python will display when you type the following into the interpreter. Then try it to check your answers.

```
>>> x = [1, 2, 3, 4]
>>> x[1:3]
_____

>>> x[:2]
_____

>>> x[1:]
_____

>>> x[-2:3]
_____
```

### Toggle Solution

As you may have noticed, Python has a convenient notation for *slicing* to retrieve part of a list. Specifically, we can write `[start:stop]` to slice a list with two integers.

- `start` denotes the index for the beginning of the slice
- `stop` denotes the index for the end of the slice

Using negative indices for `start` and `end` behaves in the same way as indexing into negative indices. In addition, slicing a list *creates a new* list, without modifying the original list.

### Question 3: What would Python print? List operations

Predict what Python will display when you type the following into the interpreter. Then try it to check your answers.

```
>>> y = [1]
```

```
>>> len(y)
```

```
_____
```

```
>>> 1 in y
```

```
_____
```

```
>>> y + [2, 3]
```

```
_____
```

```
>>> [0] + y
```

```
_____
```

```
>>> y * 3
```

```
_____
```

```
>>> z = [[1, 2], [3, 4, 5]]
```

```
>>> len(z)
```

```
_____
```

[Toggle Solution](#)

#### Question 4: Fill in the blanks

For each of the following, use element selection to get the number 7 from the particular list in the doctest. Don't worry about making this work for all lists.

[Toggle Solution](#)

```
def get_seven_a(x):
    """
    >>> x = [1, 3, [5, 7], 9]
    >>> get_seven_a(x)
    7
    """
    """ *** YOUR CODE HERE *** """
    return _____

def get_seven_b(x):
    """
    >>> x = [[7]]
    >>> get_seven_b(x)
    7
    """
    """ *** YOUR CODE HERE *** """
    return _____

def get_seven_c(x):
    """
    >>> x = [1, [2, [3, [4, [5, [6, [7]]]]]]]
    >>> get_seven_c(x)
    7
    """
    """ *** YOUR CODE HERE *** """
    return _____
```

## Question 5: Reverse (recursively)

Write a function `reverse_recursive` that takes a list and returns a new list that is the reverse of the original. Use recursion! You may also use slicing notation.

### [Toggle Solution](#)

```
def reverse_recursive(lst):
    """Returns the reverse of the given list.

    >>> reverse_recursive([1, 2, 3, 4])
    [4, 3, 2, 1]
    """
    """ *** YOUR CODE HERE *** """
```

## Question 6: Merge

Write a function `merge` that takes 2 *sorted* lists `lst1` and `lst2`, and returns a new list that

contains all the elements in the two lists in sorted order.

### [Toggle Solution](#)

```
def merge(lst1, lst2):
    """Merges two sorted lists recursively.

    >>> merge([1, 3, 5], [2, 4, 6])
    [1, 2, 3, 4, 5, 6]
    >>> merge([], [2, 4, 6])
    [2, 4, 6]
    >>> merge([1, 2, 3], [])
    [1, 2, 3]
    >>> merge([5, 7], [2, 4, 6])
    [2, 4, 5, 6, 7]
    """
    """*** YOUR CODE HERE ***"""
```

## List Comprehensions

---

List comprehensions are a compact and powerful way of creating new lists out of sequences. Let's work with them directly:

```
>>> [i**2 for i in [1, 2, 3, 4] if i%2 == 0]
[4, 16]
```

is equivalent to

```
>>> lst = []
>>> for i in [1, 2, 3, 4]:
...     if i % 2 == 0:
...         lst += [i**2]
>>> lst
[4, 16]
```

The general syntax for a list comprehension is

```
[<expression> for <element> in <sequence> if <conditional>]
```

The syntax is designed to read like English: "Compute the expression for each element

in the sequence if the conditional is true."

## Question 7: What Would Python Print?

What would Python print? Try to figure it out before you type it into the interpreter!

```
>>> [x*x for x in range(5)]
```

```
_____
```

```
>>> [n for n in range(10) if n % 2 == 0]
```

```
_____
```

```
>>> ones = [1 for i in ["hi", "bye", "you"]]
```

```
>>> ones + [str(i) for i in [6, 3, 8, 4]]
```

```
_____
```

```
>>> [i+5 for i in [n for n in range(1,4)]]
```

```
_____
```

[Toggle Solution](#)

## Question 8: Perfect squares

Implement the function `squares`, which takes in a list of positive integers, and returns a new list which contains only elements of the original list that are perfect squares. Use a list comprehension.

[Toggle Solution](#)

```

from math import sqrt

def is_square(n):
    return float(sqrt(n)) == int(sqrt(n))

def squares(seq):
    """Returns a new list containing elements of the original list that are
    perfect squares.

    >>> seq = [49, 8, 2, 1, 102]
    >>> squares(seq)
    [49, 1]
    >>> seq = [500, 30]
    >>> squares(seq)
    []
    """
    """ YOUR CODE HERE """
    return _____

```

## Extra Questions

---

Questions in this section are not required for submission. However, we encourage you to try them out on your own time for extra practice.

### Question 9: Reverse (iteratively)

Write a function `reverse_iter` that takes a list and returns a new list that is the reverse of the original. Use iteration! You may also use slicing notation.

[Toggle Solution](#)

```

def reverse_iter(lst):
    """Returns the reverse of the given list.

    >>> reverse_iter([1, 2, 3, 4])
    [4, 3, 2, 1]
    """
    """ YOUR CODE HERE """

```

### Question 10: Mergesort

[Mergesort](#) is a type of sorting algorithm. It follows a naturally recursive procedure:



- Break the input list into equally-sized halves
- Recursively sort both halves
- Merge the sorted halves.

Using your `merge` function from the previous question, implement `mergesort`.

*Challenge:* Implement mergesort itself iteratively, without using recursion.

### [Toggle Solution](#)

```
def mergesort(seq):
    """Mergesort algorithm.

    >>> mergesort([4, 2, 5, 2, 1])
    [1, 2, 2, 4, 5]
    >>> mergesort([])    # sorting an empty list
    []
    >>> mergesort([1])   # sorting a one-element list
    [1]
    """
    """*** YOUR CODE HERE ***"""
```

## Question 11: Coordinates

Implement a function `coords`, which takes a function, a sequence, and an upper and lower bound on output of the function. `coords` then returns a list of x, y coordinate pairs (lists) such that:

- Each pair contains `[x, fn(x)]`
- The x coordinates are the elements in the sequence
- Only pairs whose y coordinate is within the upper and lower bounds are included

See the doctests for examples.

One other thing: your answer can only be *one line long*. You should make use of list comprehensions!

### [Toggle Solution](#)

```
def coords(fn, seq, lower, upper):
    """
    >>> seq = [-4, -2, 0, 1, 3]
    >>> fn = lambda x: x**2
    >>> coords(fn, seq, 1, 9)
    [[-2, 4], [1, 1], [3, 9]]
    """
    """ YOUR CODE HERE """
    return _____
```

## Question 12: Deck of cards

Write a list comprehension that will create a deck of cards, given a list of suits and a list of numbers. Each element in the list will be a card, which is represented by a 2-element list of the form [suit, number].

### [Toggle Solution](#)

```
def deck(suits, numbers):
    """Creates a deck of cards (a list of 2-element lists) with the given
    suits and numbers. Each element in the returned list should be of the form
    [suit, number].

    >>> deck(['S', 'C'], [1, 2, 3])
    [['S', 1], ['S', 2], ['S', 3], ['C', 1], ['C', 2], ['C', 3]]
    >>> deck(['S', 'C'], [3, 2, 1])
    [['S', 3], ['S', 2], ['S', 1], ['C', 3], ['C', 2], ['C', 1]]
    >>> deck([], [3, 2, 1])
    []
    >>> deck(['S', 'C'], [])
    []
    """
    """ YOUR CODE HERE """
    return _____
```

## Question 13: Adding matrices

To practice, write a function that adds two matrices together using list comprehensions. The function should take in two 2D lists of the same dimensions. Try to implement this in one line!

### [Toggle Solution](#)

```
def add_matrices(x, y):
    """
    >>> matrix1 = [[1, 3],
    ...           [2, 0]]
    >>> matrix2 = [[-3, 0],
    ...           [1, 2]]
    >>> add_matrices(matrix1, matrix2)
    [[-2, 3], [3, 2]]
    """
    """ *** YOUR CODE HERE *** """
    return _____
```