Scientific Programming with Python

Collections: Lists, Tuples, Aliasing

UNIVERSITY of
HOUSTON
DIVISION OF RESEARCH
HEWLETT PACKARD ENTERPRISE DATA SCIENCE INSTITUTE

Collections

collection are data type composed of smaller pieces

- **Strings**

- **Lists**

- Tuples

- **Dictionaries**

- Sets

Lists

- A list is a linear data structure, meaning that its elements have a linear ordering.

- List is a sequential collection of data objects

- Items in a list can be accessed by indexing, and sublists can be accessed by slicing.

- Lists are mutable; individual items or entire slices can be replaced through assignment statements.

- Lists support a number of convenient and frequently used methods.

- Lists will grow and shrink as needed. With append, access insert, update, concatenate and delete operations.

Lists: Review

- A list is a linear data structure, meaning that its elements have a linear ordering.

- An example of a list is sequence of daily temperatures for a given week:

0:	68.8
1:	70.2
2:	67.2
3:	71.8
4:	73.2
5:	75.6
6:	74.0

The location at index 0 stores the temperature for Sunday, the location at index 1 stored the temperature for Monday, and so on. It is customary in programming languages to begin numbering sequences of items with an index value of 0 rather than 1. **This is referred to as** *zero-based indexing*.

Lists: Review

- A **list traversal** is a means of accessing, one-by-one, each element of a list.

- List traversal may be used, for example, to:

  - search for a particular item in a list

  - add up all the elements of a list

Lists: Review

- List can be accessed in loops

- Directly with items

```
listofnames = ['Mary', 'Susan', 'John']
for name in listofnames:
            …
```

- Or through indices

```
        for i in range(len(listofnames)):
                name = listofnames[i]

            …
```

Lists (Sequences) in Python

- A **list** in Python is a mutable, linear data structure of variable length, allowing mixed-type elements.

- By *mutable* is meant that the contents of the list may be altered. **Lists in Python use zero-based indexing**. Thus, all lists have index values 0..n-1, where n is the number of elements in the list.

Lists (Sequences) in Python

- **Lists are denoted by a** comma-separated list of elements within square brackets,

    [1, 2, 3]   ['one', 'two', 'three']   ['apples', 50, True]

- An empty list is denoted by an empty pair of square brackets, []. Elements of a list are accessed by use of an index value within square brackets,

 lst = [1, 2, 3]

    lst[0] → 1   access of first element
    lst[1] → 2   access of second element
    lst[2] → 3   access of third element

- For example, the following prints the first element of list lst,

  print (lst[0])

- The elements of lst can be summed as follows,

  sum = lst[0] + lst[2] + lst[2]

- To update,   lst[2] = 4   replacement of 3 with 4
-
- To delete,   del lst[2]   removal of 4 at index 2
- To insert,   lst.insert(1,3) insertion of 3 at index 1
- To append, lst.append(4) appends 4 to end of list

Operation	fruit = ['banana', 'apple, 'cherry']	
Replace	`fruit[2] = 'coconut'`	`['banana', 'apple', 'coconut']`
Delete	`del` fruit[0]	`['apple', 'cherry']`
Insert	`fruit.insert(2, 'pear')`	`['banana', 'apple', 'pear', 'cherry']`
Append	`fruit.append('peach')`	`['banana', 'apple', 'cherry', 'peach']`
Sort	`fruit.sort()`	`['apple', 'banana', 'cherry']`
Reverse	`fruit.reverse()`	`['cherry', 'apple', 'banana']`

Let's Try It

From the Python Shell, enter the following and observe the results.

From the Python Shell, enter the following and observe the results.

```
>>> lst = [10, 20, 30]              >>> del lst[2]
>>> lst                             >>> lst
???                                 ???

>>> lst[0]                          >>> lst.insert(1, 15)
???                                 >>> lst
                                    ???

>>> lst[0] = 5
>>> lst                             >>> lst.append(40)
???                                 >>> lst
                                    ???
```

Tuples

- A **tuple** is an *immutable* linear data structure. Thus, in contrast to lists, once a tuple is defined it cannot be altered. Otherwise, tuples and lists are essentially the same.

- To distinguish tuples from lists, tuples are denoted by parentheses instead of square brackets as given below,

nums = (10, 20, 30)

student = ('John Smith', 48, 'Computer Science', 3.42)

One element Tuple

- Another difference of tuples and lists is that tuples of one element must include a comma following the element. Otherwise, the parenthesized element will not be made into a tuple, as shown below,

- CORRECT   WRONG

>>> (1,)   >>> (1)
  (1)   **1**

Empty Tuple, and Accessing Tuples

- An empty tuple is represented by a set of empty parentheses, ().

- The elements of tuples are accessed the same as lists, with square brackets,

nums = (10, 20, 30)
student = ('John Smith', 48, 'Computer Science', 3.42)

>>> nums[0]
 10
>>> student[0]
'John Smith'

- **Any attempt to alter a tuple is invalid.** Thus, delete, update, insert and append operations are not defined on tuples.

Sequences

- A **sequence** in Python is a **linearly-ordered set of elements accessed by index number**.

- Lists, tuples and strings are all sequences. Strings, like tuples, are immutable, therefore they cannot be altered.

Operation		String s = 'hello' w = '!'	Tuple s = (1,2,3,4) w = (5,6)	List s = [1,2,3,4] w = [5,6]
Length	`len(s)`	5	4	4
Select	`s[0]`	'h'	1	1
Slice	`s[1:4]` `s[1:]`	'ell' 'ello'	(2, 3, 4) (2, 3 ,4)	[2 ,3, 4] [2, 3, 4]
Count	`s.count('e')` `s.count(4)`	1 --	-- 1	-- 1
Index	`s.index('e')` `s.index(3)`	1 --	-- 2	-- 2
Membership	`'h' in s`	True	False	False
Concatenation	`s + w`	'hello!'	(1, 2, 3, 4, 5, 6)	[1, 2, 3, 4, 5, 6]
Minimum Value	`min(s)`	'e'	1	1
Maximum Value	`max(s)`	'o'	4	4
Sum	`sum(s)`	n/a	10	10

Let's Try It

From the Python Shell, enter the following and observe the results.

```
>>> s = 'coconut'        >>> s = (10, 30, 20, 10)     >>> s = [10, 30 ,20, 10]
>>> s[4:7]               >>> s[1:3]                   >>> s[1:3]
???                      ???                          ???

>>> s.count('o')         >>> s.count(10)              >>> s.count(10)
???                      ???                          ???

>>> s.index('o')         >>> s.index(10)              >>> s.index(10)
???                      ???                          ???

>>> s + ' juice'         >>> s + (40, 50)             >>> s + [40, 50]
???                      ???                          ???
```

Nested Lists

- **Lists and tuples can contain elements of any type**, including other sequences.

- Thus, **lists and tuples can be nested** to create arbitrarily complex data structures

```
class_grades = [ [85, 91, 89], [78, 81, 86], [62, 75, 77], ...]
```

This list stores three exam grades for each student.

```
class_grades[0] equals [85, 91, 89]
class_grades[1] equals [78, 81, 86]
```

To access the first exam grade of the first student in the list,

```
        student1_grades = class_grades[0]   →  [85, 91, 89]
        student1_exam1 = student1_grades[0]  →  85
```

OR

```
        class_grades[0][0]  →    85
```

To calculate the average on the first exam for group of students, a while loop can be constructed that iterates over the first grade of each student's list of grades,

```
sum = 0
k = 0

while k < len(class_grades):
    sum = sum + class_grades[k][0]
    k = k + 1

average_exam1 = sum / float(len(grades))
```

To produce a new list names `exam_avgs` containing the exam average for each student in the class,

```
exam_avgs = []
k = 0

while k < len(class_grades):
    avg = (class_grades[k][0] + \
            class_grades[k][1] + \
            class_grades[k][2]) / 3.0

    exam_avgs.append(avg)
     k = k + 1
```

Let's Try It

From the Python Shell, enter the following and observe the results.

```
>>> lst = [[1, 2 ,3], [4, 5, 6], [7, 8, 9]]
>>> lst[0]
???
>>> lst[0][1]
???
```

```
>>> for k in lst:
        print k
???
>>> for k in lst[0]:
        print k
```

Storage of Objects

`x = 'myname'`

`y = 'myname'`

How many copies of the string in memory?

`x == y`   (The values of the variables is same)

`x is y?`   (it is the same object in memory)

Storage of List Objects

```
lx   = [1, 2, 3]
ly   = [1, 2, 3]
```
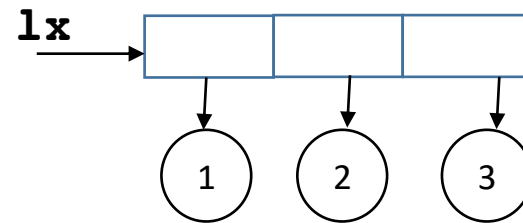
Storage of List Objects

```
lx  = [1, 2, 3]
ly = lx        (Aliasing)
lz = lx [:]    (Cloning)
```
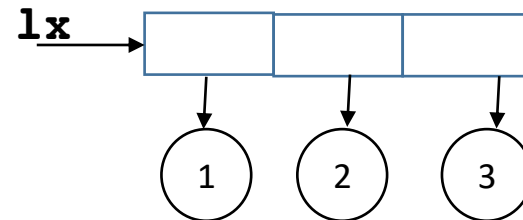
Passing Lists as Parameters

```
def listfun (lpar):
    lpar[1]=100
    return lpar
lx  = [1, 2, 3]
ly = listfun(lx)
print (ly)
```

ly

1   100   3

print (lx)

lx

1   ?   3
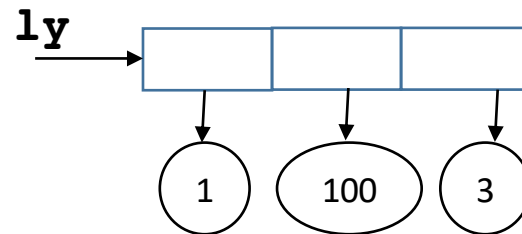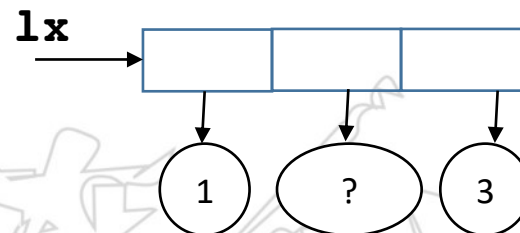
Pure Functions

Function Side Effects: Changes to variable values in the calling program as a result of function call

A pure function has no side effects!

(Functions should preferably not have side effects but there are exceptions)

Pure Function?

```python
def listfun (lpar):
    lpar[1]=100
    return lpar
```

```python
lx  = [1, 2, 3]
ly = listfun(lx)
print (ly)
print (lx)
```
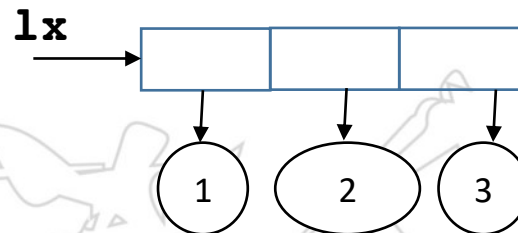
Pure Function?

```
def listfun (lpar):
    loclist = lpar(:)
    loclist[1]=100
    return loclist


lx  = [1, 2, 3]
ly = listfun(lx)
print (ly)
print (lx)
```

lx

Self-Test Questions

1. For nums = [10,30,20,40], what does the following for loop output?

```
for k in nums:
    print(k)
```

(a) 10   (b) 10   (c) 10
    20   30   30
    30   20   20
    40   40

2. For nums = [10, 30, 20, 40], what does the following for loop output?

```
for k in range(1, 4):
    print ( nums[k] )
```

(a) 10   (b) 30   (c) 10
    30   20   30
    20   40   20
                                40

3. For fruit = 'strawberry', what does the following for loop output?

```
for k in range(0, len(fruit), 2):
    print ( fruit[k] )
```

(a) srwer   (b) tabry

4. For nums = [12, 4, 11, 23, 18, 41, 27], what is the value of k when the while loop terminates?

```
k = 0
while k < len(nums) and nums[k] != 18:
    k = k + 1
```

(a) 3   (b) 4   (c) 5

ANSWERS: 1.

Self-Test Questions

1. For nums = [10,30,20,40], what does the following for loop output?

```
for k in nums:
    print(k)
```

(a) 10
20
30
40

(b) 10
30
20
40

(c) 10
30
20

2. For nums = [10, 30, 20, 40], what does the following for loop output?

```
for k in range(1, 4):
    print ( nums[k] )
```

(a) 10
30
20

(b) 30
20
40

(c) 10
30
20
40

3. For fruit = 'strawberry', what does the following for loop output?

```
for k in range(0, len(fruit), 2):
    print ( fruit[k] )
```

(a) srwer

(b) tabry

4. For nums = [12, 4, 11, 23, 18, 41, 27], what is the value of k when the while loop terminates?

```
k = 0
while k < len(nums) and nums[k] != 18:
    k = k + 1
```

(a) 3   (b) 4   (c) 5

ANSWERS: 1. (b)

Self-Test Questions

1. For `nums = [10,30,20,40]`, what does the following for loop output?

```
for k in nums:
    print(k)
```

  (a) 10   (b) 10   (c) 10
       20   30   30
       30   20   20
       40   40

2. For `nums = [10, 30, 20, 40]`, what does the following for loop output?

```
for k in range(1, 4):
    print ( nums[k] )
```

  (a) 10   (b) 30   (c) 10
       30   20   30
       20   40   20
                                40

3. For `fruit = 'strawberry'`, what does the following for loop output?

```
for k in range(0, len(fruit), 2):
    print ( fruit[k] )
```

  (a) srwer   (b) tabry

4. For `nums = [12, 4, 11, 23, 18, 41, 27]`, what is the value of k when the while loop terminates?

```
k = 0
while k < len(nums) and nums[k] != 18:
    k = k + 1
```

  (a) 3   (b) 4   (c) 5

ANSWERS: 1. (b), 2. (b),

Self-Test Questions

1. For `nums = [10,30,20,40]`, what does the following for loop output?

```
for k in nums:
    print(k)
```

(a) 10   (b) 10   (c) 10
    20   30   30
    30   20   20
    40   40

2. For `nums = [10, 30, 20, 40]`, what does the following for loop output?

```
for k in range(1, 4):
    print( nums[k] )
```

(a) 10   (b) 30   (c) 10
    30   20   30
    20   40   20
                         40

3. For `fruit = 'strawberry'`, what does the following for loop output?

```
for k in range(0, len(fruit), 2):
    print( fruit[k] )
```

(a) srwer   (b) tabry

4. For `nums = [12, 4, 11, 23, 18, 41, 27]`, what is the value of k when the while loop terminates?

```
k = 0
while k < len(nums) and nums[k] != 18:
    k = k + 1
```

(a) 3   (b) 4   (c) 5

ANSWERS: 1. (b), 2. (b), 3. (a), 4. ▮

Self-Test Questions

1. For nums = [10,30,20,40], what does the following for loop output?

```
for k in nums:
    print(k)
```

(a) 10	(b) 10	(c) 10
20	30	30
30	20	20
40	40	

2. For nums = [10, 30, 20, 40], what does the following for loop output?

```
for k in range(1, 4):
    print ( nums[k] )
```

(a) 10	(b) 30	(c) 10
30	20	30
20	40	20
		40

3. For fruit = 'strawberry', what does the following for loop output?

```
for k in range(0, len(fruit), 2):
    print ( fruit[k] )
```

(a) srwer   (b) tabry

4. For nums = [12, 4, 11, 23, 18, 41, 27], what is the value of k when the while loop terminates?

```
k = 0
while k < len(nums) and nums[k] != 18:
    k = k + 1
```

(a) 3   (b) 4   (c) 5

ANSWERS: 1. (b), 2. (b), 3. (a), 4. (b)

UNIVERSITY of
HOUSTON
DIVISION OF RESEARCH
HEWLETT PACKARD ENTERPRISE DATA SCIENCE INSTITUTE

List Comprehensions

- The range function allows for the generation of sequences of integers in fixed increments.

- **List comprehensions** in Python can be used to generate more varied sequences.

List Comprehensions

Example List Comprehensions	Resulting List
(a) `[x**2 for x in [1, 2, 3]]`	`[1, 4, 9]`
(b) `[x**2 for x in range(5)]`	`[0, 1, 4, 9, 16]`
(c) `nums = [-1, 1, -2, 2, -3, 3, -4, 4]` `[x for x in nums if x >= 0]`	`[1, 2, 3, 4]`
(d) `[ord(ch) for ch in 'Hello']`	`[72, 101, 108, 108, 111]`

(e) `vowels = ('a', 'e', 'i', 'o', 'u')`
`w = 'Hello'`
`[ch for ch in w if ch in vowels]`
	`['e', 'o']`

Let's Try It

From the Python Shell, enter the following and observe the results.

```
>>> temperatures = [88, 94, 97, 89, 101, 98, 102, 95, 100]
>>> [t for t in temperatures if t >= 100]
???

>>> [(t - 32) * 5/9.0 for t in temps]
???
```

Classwork

- Write a program to take a number K as input, reads K names (one at a time), store them in a list, and then print them in reverse (prefer not to use the reverse method)