BIOS6642

Lists

Fuyong Xing

Department of Biostatistics and Informatics

Colorado School of Public Health

University of Colorado Anschutz Medical Campus

Outline

- **List generation**
- List indexing and slicing
- List traversal
- List comprehensions
- Lists as function arguments
- List and strings
- List methods
- Multidimensional list

List generation

- A list is a sequence of objects, i.e., an ordered collection of objects.
- The elements of a list appear within square brackets, [ ], and commas separate the elements.

Codes

```
lst = [2, -3, 0, 4, -1]   # Assign the list
print([2, -3, 0, 4, -1])  # Print a literal list
print(lst)                # Print a list via a variable
```

Output

```
[2, -3, 0, 4, -1]
[2, -3, 0, 4, -1]
```

List generation

- The elements of a list do not need to have the same type.

Codes

```
collection = [24.2, 4, 'word', print, 19, -0.03, 'end']
print(collection)
```

Output

```
[24.2, 4, 'word', <built-in function print>, 19, -0.03, 'end']
```

Codes

```
col = [23, [9.3, 11.2, 99.0], [23], [], 4, [0, 0]]
print(col)
```

Output

```
[23, [9.3, 11.2, 99.0], [23], [], 4, [0, 0]]
```

List generation

- List concatenation

```
>>> a = [2, 4, 6, 8]
>>> a
[2, 4, 6, 8]
>>> a + [1, 3, 5]
[2, 4, 6, 8, 1, 3, 5]
>>> a
[2, 4, 6, 8]
>>> a = a + [1, 3, 5]
>>> a
[2, 4, 6, 8, 1, 3, 5]
>>> a += [10]
>>> a
[2, 4, 6, 8, 1, 3, 5, 10]
>>> a += 20
```

List generation

- List concatenation

```
>>> a = [2, 4, 6, 8]
>>> a
[2, 4, 6, 8]
>>> a + [1, 3, 5]
[2, 4, 6, 8, 1, 3, 5]
>>> a
[2, 4, 6, 8]
>>> a = a + [1, 3, 5]
>>> a
[2, 4, 6, 8, 1, 3, 5]
>>> a += [10]
>>> a
[2, 4, 6, 8, 1, 3, 5, 10]
>>> a += 20
```

How about this one?

List generation

- Build a list from the *range* function

```
def main():
    a = list(range(0, 10))
    print(a)
    a = list(range(10, -1, -1))
    print(a)
    a = list(range(0, 100, 10))
    print(a)
    a = list(range(-5, 6))
    print(a)


main()
```

Codes

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
```

Output

List generation

- Build a list where all the elements are the same.

```
def main():
    a = [0] * 10
    print(a)

    a = 4 * [10, 20, 30]
    print(a)

    n = 3      # Use a variable multiplier
    a = n * ['abc', 22, 8.7]
    print(a)

main()
```

Codes

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[10, 20, 30, 10, 20, 30, 10, 20, 30, 10, 20, 30]
['abc', 22, 8.7, 'abc', 22, 8.7, 'abc', 22, 8.7]
```

Output

Outline

- List generation
- **List indexing and slicing**
- List traversal
- List comprehensions
- Lists as function arguments
- List and strings
- List methods
- Multidimensional list

List indexing and slicing

- Access a list element with an index

Codes

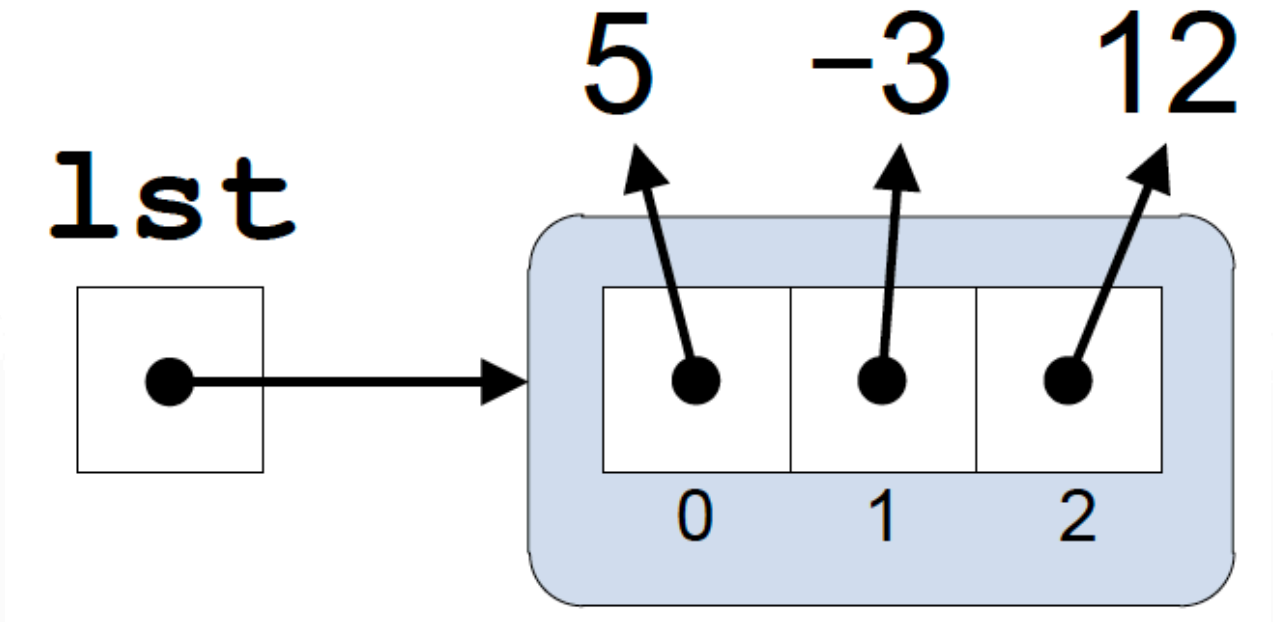
```
collection = [24.2, 4, 'word', print, 19, -0.03, 'end']
print(collection[0])
print(collection[1])
print(collection[2])
print(collection[3])
```

Output

```
24.2
4
word
<built-in function print>
```

List indexing and slicing

- *lst = [5, -3, 12]*



List indexing and slicing

- Lists are mutable.

Codes

```
lst = [2, -3, 0, 4, -1]    # Assign the list
lst[0] = 5                 # Make the first element 5
print(lst[1])              # Print the second element
lst[4] = 12                # Make the last element 12
print(lst)                 # Print a list variable
print([10, 20, 30][1])     # Print second element of literal list
```

Output

```
-3
[5, -3, 0, 4, 12]
20
```

List indexing and slicing

- Lists are mutable.

Codes

```
a = [10, 20, 30, 40]
b = a
print('a =', a)
print('b =', b)
b[2] = 35
print('a =', a)
print('b =', b)
```

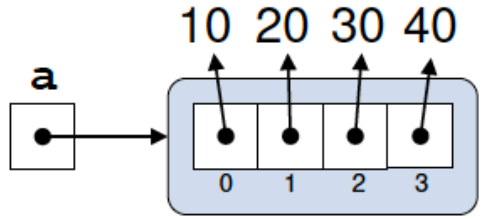
Output

```
a = [10, 20, 30, 40]
b = [10, 20, 30, 40]
a = [10, 20, 35, 40]
b = [10, 20, 35, 40]
```

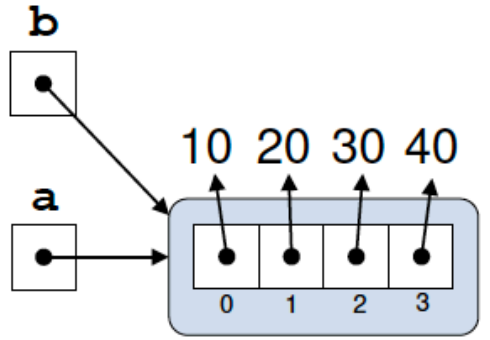
List indexing and slicing

- State diagram

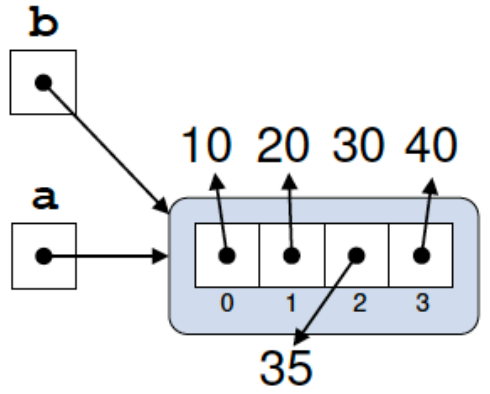
```
a = [10, 20, 30, 40]
```




```
b = a
```




```
b[2] = 35
```



List indexing and slicing

- Example (compared with the previous one)

Codes

```
a = [10, 20, 30, 40]
b = [10, 20, 30, 40]
print('a =', a)
print('b =', b)
b[2] = 35
print('a =', a)
print('b =', b)
```

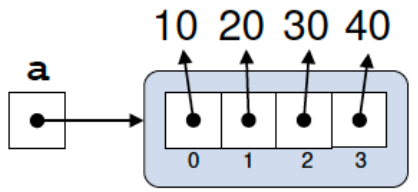
Output

```
a = [10, 20, 30, 40]
b = [10, 20, 30, 40]
a = [10, 20, 30, 40]
b = [10, 20, 35, 40]
```

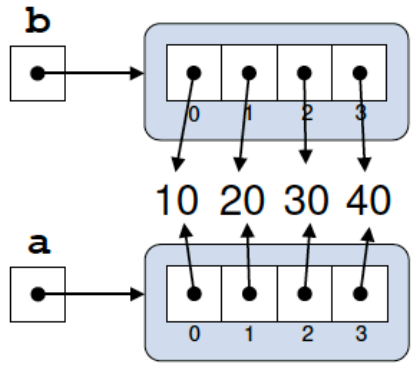
List indexing and slicing

- State diagram

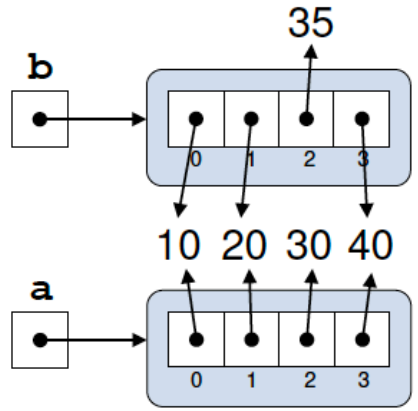
```
a = [10, 20, 30, 40]
```




```
b = [10, 20, 30, 40]
```




```
b[2] = 35
```



List indexing and slicing

- Access list elements with negative indices

```
def main():
    data = [10, 20, 30, 40, 50, 60]

    # Print the individual elements with negative indices
    print(data[-1])
    print(data[-2])
    print(data[-3])
    print(data[-4])
    print(data[-5])
    print(data[-6])

main()  # Execute main
```

Codes

```
60
50
40
30
20
10
```

Output

List indexing and slicing

- List slicing

$$list\ [\ begin\ :\ end\ :\ step\ ]$$

- *list* is a list.
- *begin* is an integer representing the starting index of a subsequence of the list.
- *end* is an integer that is one larger than the index of the last element in a subsequence of the list.
- *step* is an integer that specifies the stride size through the list.

List indexing and slicing

- List slicing

```
lst = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]
print(lst)          #  [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]
print(lst[0:3])     #  [10, 20, 30]
print(lst[4:8])     #  [50, 60, 70, 80]
print(lst[2:5])     #  [30, 40, 50]
print(lst[-5:-3])   #  [80, 90]
print(lst[:3])      #  [10, 20, 30]
print(lst[4:])      #  [50, 60, 70, 80, 90, 100, 110, 120]
print(lst[:])       #  [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]
print(lst[-100:3])  #  [10, 20, 30]
print(lst[4:100])   #  [50, 60, 70, 80, 90, 100, 110, 120]
print(lst[2:-2:2])  #  [30, 50, 70, 90]
print(lst[::2])     #  [10, 30, 50, 70, 90, 110]
print(lst[::-1])    #  [120, 110, 100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
```

List indexing and slicing

- List slicing (assume ***a*** is a list)
  - ***a[:]*** is a copy of the entire list, ***a***.
  - ***a[::-1]*** is a copy of list, ***a***, with all elements appearing in the reverse order.
- Note ***a[0]*** and ***a[0:1]*** are different.

```
>>> a = [34, -19, 20, 8, 12]
>>> print(a)
[34, -19, 20, 8, 12]
>>> print(a[0:1])
[34]
>>> print(a[0])
34
```

List indexing and slicing

- Slice assignment

Codes

```
lst = [10, 20, 30, 40, 50, 60, 70, 80]
print(lst)                    # Print the list
lst[2:5] = ['a', 'b', 'c']    # Replace [30, 40, 50] segment with ['a', 'b', 'c']
print(lst)
print('==================')
lst = [10, 20, 30, 40, 50, 60, 70, 80]
print(lst)                    # Print the list
lst[2:6] = ['a', 'b']    # Replace [30, 40, 50, 60] segment with ['a', 'b']
print(lst)
print('==================')
lst = [10, 20, 30, 40, 50, 60, 70, 80]
print(lst)           # Print the list
lst[2:2] = ['a', 'b', 'c']    # Insert ['a', 'b', 'c'] segment at index 2
print(lst)
print('==================')
lst = [10, 20, 30, 40, 50, 60, 70, 80]
print(lst)      # Print the list
lst[2:5] = []    # Replace [30, 40, 50] segment with [] (delete the segment)
print(lst)
```

List indexing and slicing

- Slice assignment

Output

```
[10, 20, 30, 40, 50, 60, 70, 80]
[10, 20, 'a', 'b', 'c', 60, 70, 80]
==================
[10, 20, 30, 40, 50, 60, 70, 80]
[10, 20, 'a', 'b', 70, 80]
==================
[10, 20, 30, 40, 50, 60, 70, 80]
[10, 20, 'a', 'b', 'c', 30, 40, 50, 60, 70, 80]
==================
[10, 20, 30, 40, 50, 60, 70, 80]
[10, 20, 60, 70, 80]
```

List indexing and slicing

- List element removal: using the *del* statement

```
>>> a = list(range(10, 51, 10))
>>> a
[10, 20, 30, 40, 50]
>>> del a[2]
>>> a
[10, 20, 40, 50]
```

```
>>> b = list(range(20))
>>> b
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> del b[5:15]
>>> b
[0, 1, 2, 3, 4, 15, 16, 17, 18, 19]
```

Outline

- List generation
- List indexing and slicing
- **List traversal**
- List comprehensions
- Lists as function arguments
- List and strings
- List methods
- Multidimensional list

List traversal

- List traversal with a *for* loop

```
collection = [24.2, 4, 'word', print, 19, -0.03, 'end']
for item in collection:
    print(item)      # Print each element
```

- Another way

```
collection = [24.2, 4, 'word', print, 19, -0.03, 'end']
for i in range(len(collection)):     # Not the preferred way to traverse a list
    print(collection[i])      # Print each element
```

List traversal

- Print list elements in a list in the reverse order

Codes

```
nums = [2, 4, 6, 8]
#  Print last element to first (zero index) element
for i in range(len(nums) - 1, -1, -1):
    print(nums[i])
```

Output

```
8
6
4
2
```

List traversal

- Determine whether an object is an element of a list

Codes

```
lst = list(range(0, 21, 2))
for i in range(-2, 23):
    if i in lst:
        print(i, 'is a member of', lst)
    if i not in lst:
        print(i, 'is NOT a member of', lst)
```

Output

```
-2 is NOT a member of [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
-1 is NOT a member of [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
0 is a member of [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
1 is NOT a member of [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
⋮
21 is NOT a member of [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
22 is NOT a member of [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Outline

- List generation
- List indexing and slicing
- List traversal
- **List comprehensions**
- Lists as function arguments
- List and strings
- List methods
- Multidimensional list

List comprehensions

- A list comprehension consists of brackets containing an expression followed by a *for* clause, then zero or more *for* or *if* clauses.

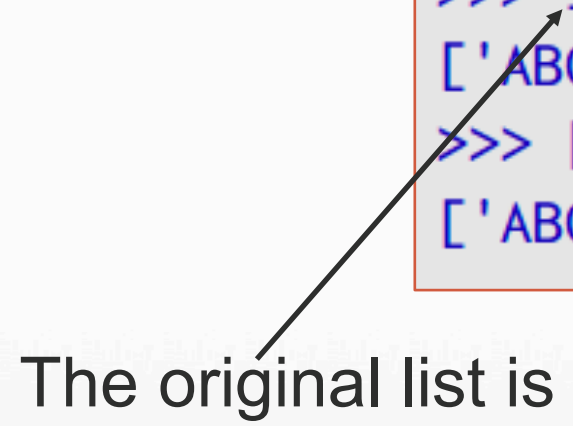
```
>>> vec = [-4, -2, 0, 2, 4]
>>> # create a new list with the values doubled
>>> [x*2 for x in vec]
[-8, -4, 0, 4, 8]
>>> # filter the list to exclude negative numbers
>>> [x for x in vec if x >= 0]
[0, 2, 4]
>>> # apply a function to all the elements
>>> [abs(x) for x in vec]
[4, 2, 0, 2, 4]
>>> # call a method on each element
>>> freshfruit = ['  banana', '  loganberry ', 'passion fruit  ']
>>> [weapon.strip() for weapon in freshfruit]
['banana', 'loganberry', 'passion fruit']
```


https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions

List comprehensions

- Examples

```
>>> lst = ['ABC', 23.4, 7, 'Wow', 16, 'xyz', 10]
>>> lst
['ABC', 23.4, 7, 'Wow', 16, 'xyz', 10]
>>> [x for x in lst if type(x) != str]
[23.4, 7, 16, 10]
>>> lst
['ABC', 23.4, 7, 'Wow', 16, 'xyz', 10]
>>> [x for x in lst if type(x) == str]
['ABC', 'Wow', 'xyz']
```



List comprehensions

- Examples

```
>>> squares = [(x, x**2) for x in range(1, 11)]
>>> squares
[(1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81), (10, 100)]
```

Outline

- List generation
- List indexing and slicing
- List traversal
- List comprehensions
- **Lists as function arguments**
- List and strings
- List methods
- Multidimensional list

Lists as function arguments

- Suppose there is a function, *delete_head,* which takes a list as input.

```
def delete_head(t):
    del t[0]
```

- Calling the function will remove the first element of the list:

```
>>> letters = ['a', 'b', 'c']
>>> delete_head(letters)
>>> letters
['b', 'c']
```

Lists as function arguments

- The *tail* function takes a list as input.

```
def tail(t):
    return t[1:]
```

- Calling the function:

```
>>> letters = ['a', 'b', 'c']
>>> rest = tail(letters)
>>> rest
['b', 'c']
```

- Is the *letters* list modified?

Outline

- List generation
- List indexing and slicing
- List traversal
- List comprehensions
- Lists as function arguments
- **List and strings**
- List methods
- Multidimensional list

Lists and strings

- Convert a string to a list of characters

```
>>> s = 'spam'
>>> t = list(s)
>>> t
['s', 'p', 'a', 'm']
```

- Break a string into words

```
>>> s = 'pining for the fjords'
>>> t = s.split()
>>> t
['pining', 'for', 'the', 'fjords']
```

Lists and strings

- Convert a string to a list of words/substrings

```
>>> s = 'spam-spam-spam'
>>> delimiter = '-'
>>> t = s.split(delimiter)
>>> t
['spam', 'spam', 'spam']
```

- The *split* method has an optional argument that specifies what character to use as the word separator.

Lists and strings

- The *joint* method takes a list of strings and concatenates the elements.

```
>>> t = ['pining', 'for', 'the', 'fjords']
>>> delimiter = ' '
>>> s = delimiter.join(t)
>>> s
'pining for the fjords'
```

Outline

- List generation
- List indexing and slicing
- List traversal
- List comprehensions
- Lists as function arguments
- List and strings
- **List methods**
- Multidimensional list

List methods

- An list is an object. Some methods of the list object are shown below.

`list` Methods
`count` Returns the number of times a given element appears in the list. Does not modify the list.
`insert` Inserts a new element before the element at a given index. Increases the length of the list by one. Modifies the list.
`append` Adds a new element to the end of the list. Modifies the list.
`index` Returns the lowest index of a given element within the list. Produces an error if the element does not appear in the list. Does not modify the list.
`remove` Removes the first occurrence (lowest index) of a given element from the list. Produces an error if the element is not found. Modifies the list if the item to remove is in the list.
`reverse` Physically reverses the elements in the list. The list is modified.

List methods

- Add a new element to the end of a list

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> t
['a', 'b', 'c', 'd']
```

- Sort the elements of the list

```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> t
['a', 'b', 'c', 'd', 'e']
```

List methods

- Example

The ***reserved*** function returns an iterator, which accesses the given sequence in the reverse order.

Codes

```
lst = [1, 2, 3, 4, 5, 6, 7]
print("---- Original list ----")
print("lst =", lst)
print("---- reversed function----")
obj1 = reversed(lst)
print("lst =", lst)
print("obj1 =", obj1)
print("---- Slice ----")
obj2 = lst[::-1]
print("lst =", lst)
print("obj2 =", obj2)
print("---- list.reverse method ----")
obj3 = lst.reverse()
print("lst =", lst)
print("obj3 =", obj3)
```

List methods

- Example

Output

```
---- Original list ----
lst = [1, 2, 3, 4, 5, 6, 7]
---- reversed function----
lst = [1, 2, 3, 4, 5, 6, 7]
obj1 = <list_reverseiterator object at 0x000000000281A320>
---- Slice ----
lst = [1, 2, 3, 4, 5, 6, 7]
obj2 = [7, 6, 5, 4, 3, 2, 1]
---- list.reverse method ----
lst = [7, 6, 5, 4, 3, 2, 1]
obj3 = None
```

Outline

- List generation
- List indexing and slicing
- List traversal
- List comprehensions
- Lists as function arguments
- List and strings
- List methods
- **Multidimensional lists**

Multidimensional lists

- A two-dimensional list: a list of lists

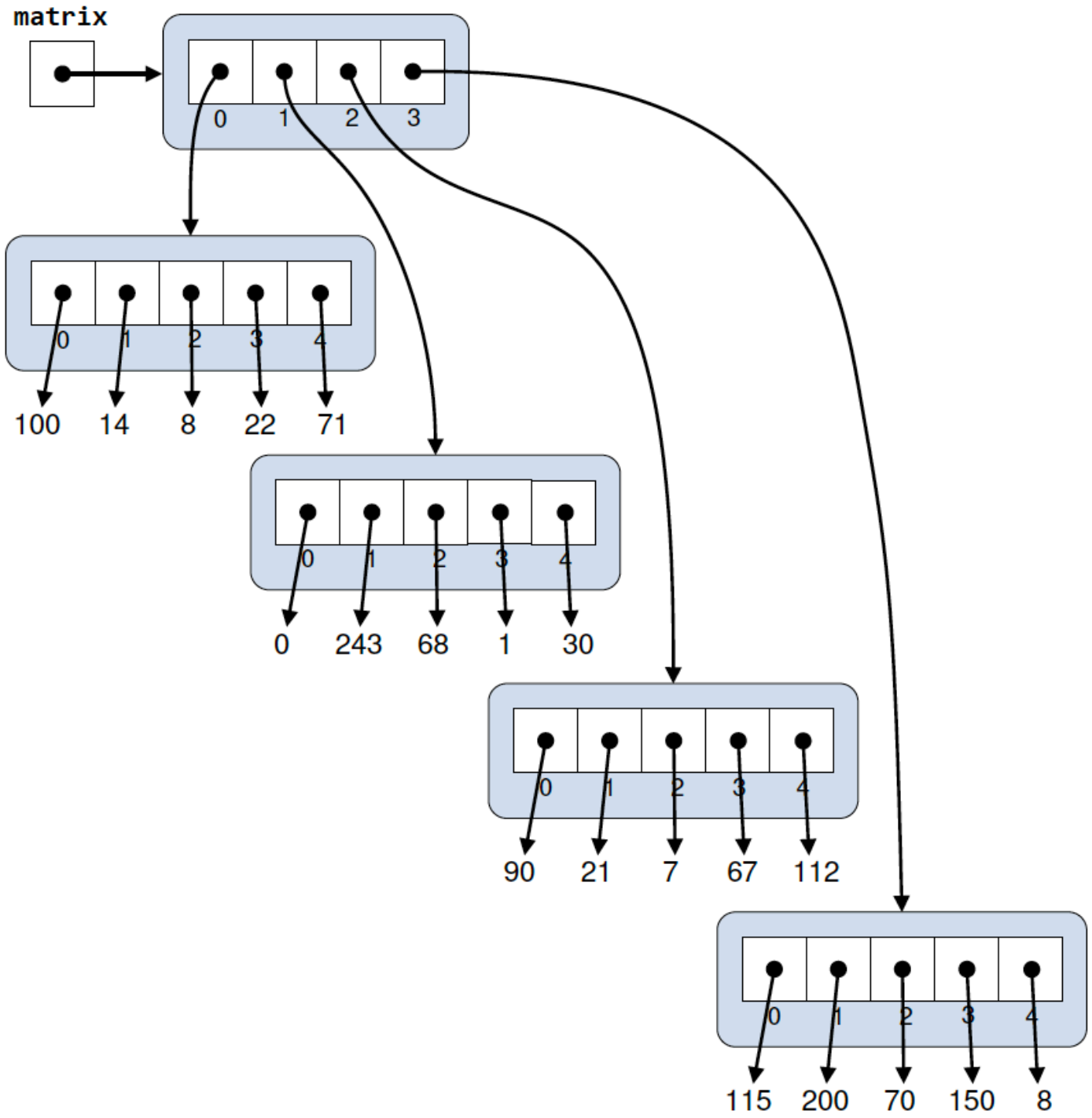
```
matrix = [[100,  14,   8,  22,  71],
          [  0, 243,  68,   1,  30],
          [ 90,  21,   7,  67, 112],
          [115, 200,  70, 150,   8]]

for row in matrix:      # Process each row
    for elem in row:    # For each element in a given row
        print('{:>4}'.format(elem), end='')
    print()
```

Codes

```
 100  14   8  22  71
   0 243  68   1  30
  90  21   7  67 112
 115 200  70 150   8
```

Output

Illustration of representation



Readings

- Think Python, chapter 10

References

- *Think Python: How to Think Like a Computer Scientist*, 2nd edition, 2015, by Allen Downey
- *Python Cookbook*, 3rd edition, by David Beazley and Brian K. Jones, 2013
- *Python for Data Analysis*, 2nd edition, by Wes McKinney, 2018
- *Fundamentals of Python Programming,* by Richard L. Halterman