Algorithmic Approaches for Biological Data, Lecture #9

Katherine St. John

City University of New York
American Museum of Natural History

22 February 2016

Outline

- Container Data Types

Outline

- Container Data Types
- Lists

Outline

- Container Data Types
- Lists
  - Accessing elements, indexing, and slicing

Outline

- Container Data Types
- Lists
  - Accessing elements, indexing, and slicing
  - Creating new lists: concatenation, repetition, appending

Outline

- Container Data Types
- Lists
  - Accessing elements, indexing, and slicing
  - Creating new lists: concatenation, repetition, appending
  - Lists and `for`-loops

Outline

- Container Data Types
- Lists
  - Accessing elements, indexing, and slicing
  - Creating new lists: concatenation, repetition, appending
  - Lists and `for`-loops
  - *Break*
  - Lists and functions

Outline

- Container Data Types
- Lists
  - Accessing elements, indexing, and slicing
  - Creating new lists: concatenation, repetition, appending
  - Lists and `for`-loops
  - *Break*
  - Lists and functions
  - Nested lists

Outline

- Container Data Types
- Lists
  - Accessing elements, indexing, and slicing
  - Creating new lists: concatenation, repetition, appending
  - Lists and `for`-loops
  - *Break*
  - Lists and functions
  - Nested lists
  - List Comprehensions

Outline

- Container Data Types
- Lists
  - Accessing elements, indexing, and slicing
  - Creating new lists: concatenation, repetition, appending
  - Lists and `for`-loops
  - *Break*
  - Lists and functions
  - Nested lists
  - List Comprehensions

Data Types

- All variables have a type that indicates what can/is stored there.

Data Types

- All variables have a type that indicates what can/is stored there.
- Examples of built-in data types thus far:

Data Types

- All variables have a type that indicates what can/is stored there.
- Examples of built-in data types thus far:
  `int`,

Data Types

- All variables have a type that indicates what can/is stored there.
- Examples of built-in data types thus far:
  `int, float,`

Data Types

- All variables have a type that indicates what can/is stored there.
- Examples of built-in data types thus far:
  `int`, `float`, `bool`,

Data Types

- All variables have a type that indicates what can/is stored there.
- Examples of built-in data types thus far:
  `int, float, bool, file,`

Data Types

- All variables have a type that indicates what can/is stored there.
- Examples of built-in data types thus far:
  `int`, `float`, `bool`, `file`, `string`,

Data Types

- All variables have a type that indicates what can/is stored there.
- Examples of built-in data types thus far:
  `int`, `float`, `bool`, `file`, `string`, `list`,

Data Types

- All variables have a type that indicates what can/is stored there.
- Examples of built-in data types thus far:
  `int`, `float`, `bool`, `file`, `string`, `list`, `None`

Data Types

- All variables have a type that indicates what can/is stored there.
- Examples of built-in data types thus far:
  `int`, `float`, `bool`, `file`, `string`, `list`, `None`
- In many programming languages, you must declare the type of a variable before using it.

Data Types

- All variables have a type that indicates what can/is stored there.
- Examples of built-in data types thus far:
  `int`, `float`, `bool`, `file`, `string`, `list`, `None`
- In many programming languages, you must declare the type of a variable before using it.
- Python has dynamic typing where the type is deduced when value assigned.

Data Types

- All variables have a type that indicates what can/is stored there.
- Examples of built-in data types thus far:
  `int`, `float`, `bool`, `file`, `string`, `list`, `None`
- In many programming languages, you must declare the type of a variable before using it.
- Python has dynamic typing where the type is deduced when value assigned.
- Can test what type Python assigned using the `type()` function.

Data Types

- All variables have a type that indicates what can/is stored there.
- Examples of built-in data types thus far:
  `int`, `float`, `bool`, `file`, `string`, `list`, `None`
- In many programming languages, you must declare the type of a variable before using it.
- Python has dynamic typing where the type is deduced when value assigned.
- Can test what type Python assigned using the `type()` function.
- *Demo*

Container Data Types

- Container data types contain multiple values.

Container Data Types

- Container data types contain multiple values.
- Examples thus far:

Container Data Types

- Container data types contain multiple values.
- Examples thus far:
  `string`,

Container Data Types

- Container data types contain multiple values.
- Examples thus far:
  `string`, `list`

Container Data Types

- Container data types contain multiple values.
- Examples thus far:
  `string`, `list`
- Other built-in container types include: tuples, dictionaries, and sets

Container Data Types

- Container data types contain multiple values.
- Examples thus far:
  `string`, `list`
- Other built-in container types include: tuples, dictionaries, and sets
- There are additional container data types used in popular modules such as `numpy`.

Lists

- A sequence of elements.

Lists

- A sequence of elements.
- Represented as a bracketed lists of comma-separated values:

Lists

- A sequence of elements.
- Represented as a bracketed lists of comma-separated values:
  `myList = [3,1,4.0,"hello",-12]`

Lists

- A sequence of elements.
- Represented as a bracketed lists of comma-separated values:
  `myList = [3,1,4.0,"hello",-12]`
- Using lists since first day of class.

Lists

- A sequence of elements.
- Represented as a bracketed lists of comma-separated values:
  `myList = [3,1,4.0,"hello",-12]`
- Using lists since first day of class.

```
for i in range(4):
    tess.forward(100)
    tess.left(90)
```

Lists

- A sequence of elements.
- Represented as a bracketed lists of comma-separated values:
  `myList = [3,1,4.0,"hello",-12]`
- Using lists since first day of class.

```
for i in range(4):
    tess.forward(100)
    tess.left(90)
```

- Like strings in many ways and many built-in functions work for either.

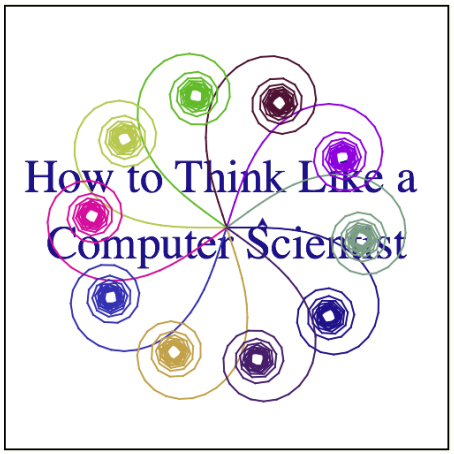
Lists

- A sequence of elements.
- Represented as a bracketed lists of comma-separated values:
  `myList = [3,1,4.0,"hello",-12]`
- Using lists since first day of class.

```
for i in range(4):
    tess.forward(100)
    tess.left(90)
```

- Like strings in many ways and many built-in functions work for either.
- Note: difference between Python 2 and Python3– can't clone or test equality of elements in Python 2 (book is written for Python 3).

In Pairs

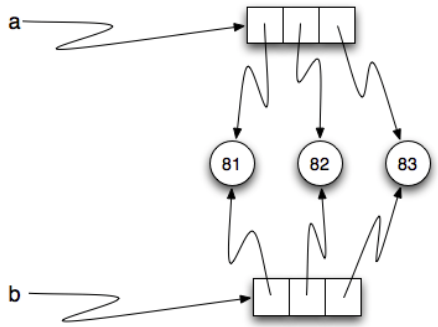


In pairs, work through:

1. *Think CS* Lists Chapter: List Length
2. *Think CS* Lists Chapter: Accessing Elements
3. *Think CS* Lists Chapter: List Membership
4. *Think CS* Lists Chapter: Concatenation and Repetition
5. *Think CS* Lists Chapter: List Slices
6. Write code that reverse a list and concatenates it to itself.

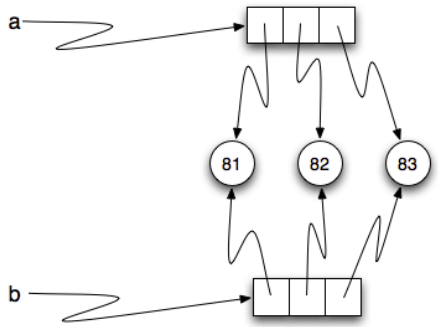
Changing Lists

- Lists are mutable.

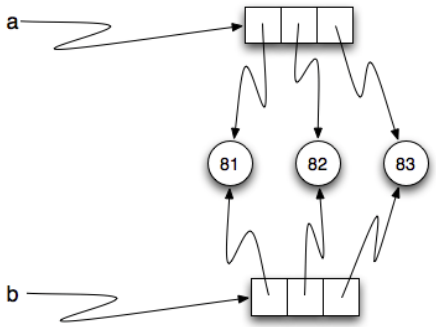


Changing Lists

- Lists are mutable.
  `myList[2] = 4.5`

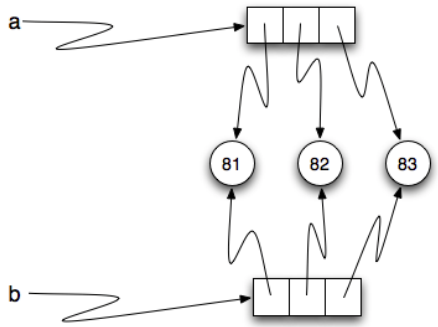


Changing Lists



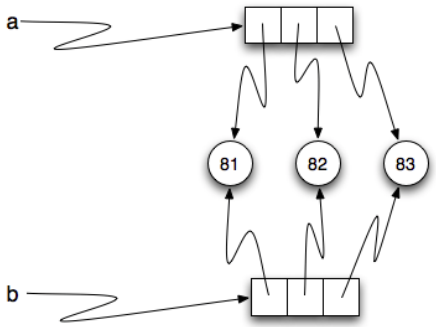
- Lists are mutable.
  `myList[2] = 4.5`
- Can delete elements:

Changing Lists



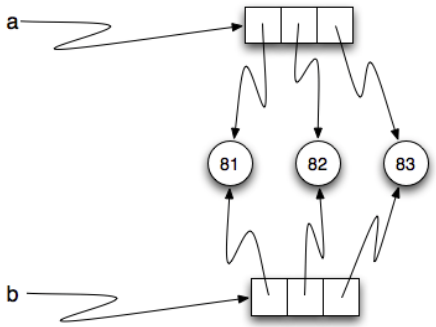
- Lists are mutable.
  `myList[2] = 4.5`
- Can delete elements:
  `del myList[2:4]`

Changing Lists



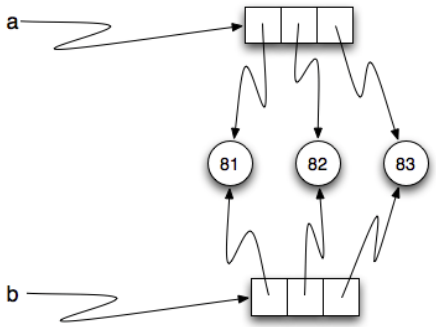
- Lists are mutable.
  `myList[2] = 4.5`
- Can delete elements:
  `del myList[2:4]`
- Elements in the list are references to values:

Changing Lists



- Lists are mutable.
  `myList[2] = 4.5`
- Can delete elements:
  `del myList[2:4]`
- Elements in the list are references to values:
  `a = [81,82,83]`
  `b = [81,82,83]`

Changing Lists

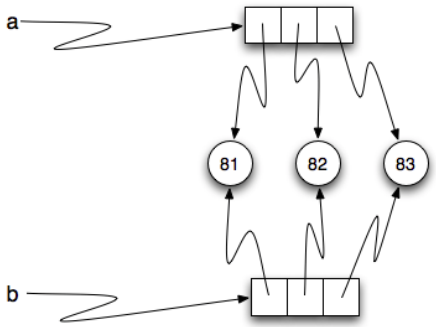


- Lists are mutable.
  `myList[2] = 4.5`
- Can delete elements:
  `del myList[2:4]`
- Elements in the list are references to values:

```
a = [81,82,83]
b = [81,82,83]

print a is b
print a == b
```

Changing Lists



- Lists are mutable.
  `myList[2] = 4.5`
- Can delete elements:
  `del myList[2:4]`
- Elements in the list are references to values:

```
a = [81,82,83]
b = [81,82,83]

print a is b
print a == b
```

- Different behavior in Python 3: equality will test elements and print True.

Lists and `for`-loops

- Commonly use lists in `for`-loops:

Lists and `for`-loops

- Commonly use lists in `for`-loops:

```
for i in range(4):
    tess.forward(100)
    tess.left(90)
```

Lists and `for`-loops

- Commonly use lists in `for`-loops:

```
for i in range(4):
    tess.forward(100)
    tess.left(90)
```

- Can also use `append()` to build up lists.

Lists and `for`-loops

- Commonly use lists in `for`-loops:

```
for i in range(4):
    tess.forward(100)
    tess.left(90)
```

- Can also use `append()` to build up lists.

```
squares = []
for i in range(-4,4):
    squares.append(i**2)
print squares
```

Accumulator Design Pattern

- "Accumulating" a value is a design approach that occurs frequently.

Accumulator Design Pattern

- "Accumulating" a value is a design approach that occurs frequently.
- Example:

```
total = 0
for i in range(11,22,2):
    total = total + i
    print i, total
```

- Simplest forms are running sums and running products.

Accumulator Design Pattern

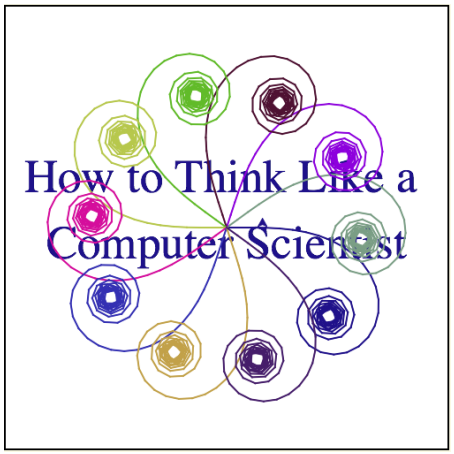
- "Accumulating" a value is a design approach that occurs frequently.
- Example:

```
total = 0
for i in range(11,22,2):
    total = total + i
    print i, total
```

- Simplest forms are running sums and running products.

	Running Sums:	Running Products:	Accumulating Strings:	Accumulating Lists:
Initialization:	s = 0	p = 1	s = ""	s = []
Update Action:	for ... s=s+newValue	for ... p=p*newValue	for ... s=s+newValue	for ... s.append(newValue)

In Pairs



In pairs, work through:

1. *Think CS* Lists Chapter: Lists are Mutable
2. *Think CS* Lists Chapter: List Deletion
3. *Think CS* Lists Chapter: Repetition & References
4. *Think CS* Lists Chapter: Lists & `for`-loops
5. Create a list that contains the first 10 cubes.

Break

Asia Collections, AMNH

Functions & Lists

- Functions can take list as parameters and return lists.

Functions & Lists

- Functions can take list as parameters and return lists.
- Recall that functions cannot alter parameters.

Functions & Lists

- Functions can take list as parameters and return lists.
- Recall that functions cannot alter parameters.
- While the list pointer cannot be changed, its elements can.

Functions & Lists

- Functions can take list as parameters and return lists.
- Recall that functions cannot alter parameters.
- While the list pointer cannot be changed, its elements can.
  (unexpected side affects that should be avoided– *pure functions* preferred.

Functions & Lists

- Functions can take list as parameters and return lists.
- Recall that functions cannot alter parameters.
- While the list pointer cannot be changed, its elements can.
  (unexpected side affects that should be avoided– *pure functions* preferred.
- *PythonTutor demo*

Functions & Lists

- Functions can take list as parameters and return lists.
- Recall that functions cannot alter parameters.
- While the list pointer cannot be changed, its elements can.
  (unexpected side affects that should be avoided– *pure functions* preferred.
- *PythonTutor demo*

Nested Lists

- Lists can be placed inside other lists.

Nested Lists

- Lists can be placed inside other lists.
- Access from outer most:

Nested Lists

- Lists can be placed inside other lists.
- Access from outer most:

```
alist = [ [4, [True, False], 6, 8], [888, 999] ]
if alist[0][1][0]:
   print(alist[1][0])
else:
   print(alist[1][1])
```

(From book, uses Python 3)

Nested Lists

- Lists can be placed inside other lists.
- Access from outer most:

```
alist = [ [4, [True, False], 6, 8], [888, 999] ]
if alist[0][1][0]:
   print(alist[1][0])
else:
   print(alist[1][1])
```

  (From book, uses Python 3)

- *PythonTutor demo*

List Comprehensions

- Shorthand way of creating lists from other lists.

List Comprehensions

- Shorthand way of creating lists from other lists.
- Examples:

List Comprehensions

- Shorthand way of creating lists from other lists.
- Examples:
  - `sq = [i**2 for i in range(10)]`

List Comprehensions

- Shorthand way of creating lists from other lists.
- Examples:
  - `sq = [i**2 for i in range(10)]`
  - `ev = [i for i in range(20) if i%2 == 0]`

List Comprehensions

- Shorthand way of creating lists from other lists.
- Examples:
  - `sq = [i**2 for i in range(10)]`
  - `ev = [i for i in range(20) if i%2 == 0]`
- General form:

```
[<expression> for <item> in <sequence> if <condition>]
```

List Comprehensions

- Shorthand way of creating lists from other lists.
- Examples:
  - `sq = [i**2 for i in range(10)]`
  - `ev = [i for i in range(20) if i%2 == 0]`
- General form:

  `[<expression> for <item> in <sequence> if <condition>]`
- *PythonTutor demo*

In Pairs

In pairs, work through:

1. What does the following print:

```
a = []
for i in range(5):
    b = []
    for j in range(5):
        b.append(i+j)
    a.append(b)
print a
print a[1][2]
```

2. What does the following print:

```
c = [k-5 for k in range(5,0,-1)]
print c
```

3. What does the following print:

```
m = sum([x for x in range(11)])/10.0
s = sum([x-m for x in range(11)])/10.0
print m,s
```

Recap

- More on lists and strings on Wednesday.

Recap

- More on lists and strings on Wednesday.
- Email lab reports to `kstjohn@amnh.org`

Recap

- More on lists and strings on Wednesday.
- Email lab reports to `kstjohn@amnh.org`
- Challenges available at `rosalind.info`