Lists, Memory Diagrams & Mutable vs. Immutable Sequences

Homogenous, heterogeneous, nested lists

Lists in which all elements have the same type are called homogenous. Most of the lists we’ll use will be homogeneous.

# List of primes less than 20
[2, 3, 5, 7, 11, 13, 17, 19]

Lists can also contain other lists as elements!

# List of string lists
[['fox', 'raccoon'], ['duck', 'raven', 'gosling'], [], ['turkey']]

Python also allows heterogeneous lists in which elements can have different types. In general, you should avoid heterogeneous lists unless you have a good reason to use them. (They make programs harder to reason about.)

[17, True, 'Wendy', None, [42, False, 'computer']]

Lists: glue for many values

# Lists returned from built-in functions and methods
oddies = range(1, 10, 2) # [1, 3, 5, 7, 9]
lyrics = 'call me on my cell'.split() # ['call', 'me', 'on', 'my', 'cell']
letters = list('happy') # ['h', 'a', 'p', 'y']

# Literal list definitions
primes = [2, 3, 5, 7, 11, 13, 17, 19]
bools = [l<2, l==2, l>2]
houses = ['Gryffindor', 'Hufflepuff', 'Ravenclaw', 'Slytherin']
strings = ['ab'] + ['cd', 'ma'*4]
counts = [1, 2, 3] + [4, 5]
animals = [['fox', 'raccoon'], ['duck', 'raven', 'gosling'], [], ['turkey']]

# A heterogeneous list
stuff = [17, True, 'foo', None, [42, False, 'bar']]

# An empty list
empty = []
How to represent list values: Memory Diagrams [0]

numbers, booleans, and None are “small enough” to fit directly in variables and list slots.

All other values are drawn outside the variable/list slot, with an arrow pointing to them.

How to represent list values: Memory Diagrams [1]

animalLists = [['fox', 'raccoon'],
['duck', 'raven', 'gosling'],
[],
['turkey']]

List indexing and slicing (review)

In[1]: houses = ['Gryffindor', 'Hufflepuff', 'Ravenclaw', 'Slytherin']
In[2]: houses[0] # List indexing
Out[2]: 'Gryffindor'
In[3]: houses[3] # List indexing
Out[3]: 'Slytherin'
In[4]: houses[4]
IndexError: list index out of range

In[5]: houses[-3]  # Negative indexing
Out[5]: 'Ravenclaw'
In[6]: houses[1:3]  # Slicing
Out[6]: ['Hufflepuff', 'Ravenclaw']
In[7]: houses[2:]  # Slicing
Out[7]: ['Ravenclaw', 'Slytherin']
In[8]: houses[:2]  # Slicing
Out[8]: ['Gryffindor', 'Hufflepuff']

Nested list indexing

animalLists = [[['fox', 'raccoon'],
['duck', 'raven', 'gosling'],
[],
['turkey']]

animalLists = [[['fox', 'raccoon'],
['duck', 'raven', 'gosling'],
[],
['turkey']]

animalLists[0][1]  # Nested list indexing
animalLists[0][1][1]  # Nested list indexing
animalLists[0][1][1][2]  # Nested list indexing

Write a 1-line Python expression to get 'raven' from animalLists.
Write a 1-line Python expression to get 'turkey' from animalLists.

Challenge: write two new expressions that also get 'raven' and 'turkey' using different indices than before.
**Lists are sequences.**

Immutable sequence operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x in seq</td>
<td>True if an item of seq is equal to x</td>
</tr>
<tr>
<td>x not in seq</td>
<td>False if an item of seq is equal to x</td>
</tr>
<tr>
<td>seq1 + seq2</td>
<td>The concatenation of seq1 and seq2</td>
</tr>
<tr>
<td>seq<em>n, n</em>seq</td>
<td>n copies of seq concatenated</td>
</tr>
<tr>
<td>seq[i]</td>
<td>i'th item of seq, where origin is 0</td>
</tr>
<tr>
<td>seq[i:j]</td>
<td>slice of seq from i to j</td>
</tr>
<tr>
<td>seq[i:j:k]</td>
<td>slice of seq from i to j with step k</td>
</tr>
<tr>
<td>len(seq)</td>
<td>length of seq</td>
</tr>
<tr>
<td>min(seq)</td>
<td>smallest item of seq</td>
</tr>
<tr>
<td>max(seq)</td>
<td>largest item of seq</td>
</tr>
</tbody>
</table>

**Lists are **mutable.**

Lists are mutable, meaning that their contents can change over time.

Lists can change in two ways:

1. The element at a given index can change over time. That is, the slot in a list at a particular index behaves as a variable, whose contents can change over time.

2. The length of a list can change over time as new slots are added or removed.

**List slot mutability** example

shoesizes = [8, 8.5, 12.5, 10]

shoesizes[3] = 11.5

**List slot mutability** larger example [0]

myList = [17, 3.141, True, None, ['I', 'am', 'Sam'], Circle(50, Point(200, 100))]

myList[3] = 'Sam'

Circle radius = 50

Point x = 200
y = 100
**List slot mutability** larger example [1]

The value in any named or numbered box can change over time. For example, the values in list slots can be changed by assignment.

\[
\begin{align*}
\text{myList}[1] &= \text{myList}[0] + 6 \\
\text{myList}[3] &= \text{myList}[0] > \text{myList}[1] \\
\text{myList}[4][1] &= 'was'
\end{align*}
\]

**append**: add a new slot to the end of a list

\[
\begin{align*}
\text{myList}.\text{append}(42) \\
\text{myList}[4].\text{append}('Adams')
\end{align*}
\]

**List Mutability**

Assigning to a list index:

\[
\begin{align*}
\text{In [ ]: numStrings} &= ['zero', 'one', 'two', 'three', 'four'] \\
\text{In [ ]: numStrings[3]} &= 'THREE' \\
\text{In [ ]: numStrings} \\
\text{Out[ ]: ['zero', 'one', 'two', 'THREE', 'four']}
\end{align*}
\]

Adding an element to the end of a list with **append**:

\[
\begin{align*}
\text{In [ ]: numStrings.append('five')} \\
\text{In [ ]: numStrings} \\
\text{Out[ ]: ['zero', 'one', 'two', 'THREE', 'four', 'five']}
\end{align*}
\]

**More list mutability**

**pop**

(remove an element from a list)

**insert**

(adding a new element to a list)

"Aliasing"

(same object stored in multiple variables and slots)
**pop**: remove slot at an index and return its value

```python
myList.pop(3)
```

**List Diagrams/Mutability 8-17**

**pop**: remove slot at an index and return its value

```python
myList.pop(3) → False  # Indices of slots after 3 are decremented
```
```
myList[3].pop(2)
```

**List Diagrams/Mutability 8-18**

**pop**: remove slot at an index and return its value

```python
myList.pop(3) → False  # Indices of slots after 3 are decremented
myList[3].pop(2) → 'Sam'
```

**List Diagrams/Mutability 8-19**

**pop**: remove slot at an index and return its value

```python
myList.pop(3) → False  # Indices of slots after 3 are decremented
myList[3].pop(2) → 'Sam'
```

**List Diagrams/Mutability 8-20**
**pop**: remove slot at an index and return its value

1. `myList.pop(3) → False`  # Indices of slots after 3 are decremented
2. `myList[3].pop(2) → 'Sam'`  # Index of previous slot 3 is decremented
3. `myList.pop()`  

---

**insert**: add a slot, add an index

1. `myList.insert(0, 98.6)`  
2. `myList.insert(0, 98.6)`  # Indices of previous slots 0 and above are incremented
**insert**: add a slot, add an index

myList.insert(0, 98.6)  # Indices of previous slots 0 and above  
                        # are incremented

myList[4].insert(2, 'not')

---

**Aliasing**: the very same object can be stored in different variables & slots

list2 = myList

---

List Diagrams/Mutability  8-25

List Diagrams/Mutability  8-26

List Diagrams/Mutability  8-27

List Diagrams/Mutability  8-28
Aliasing: the very same object can be stored in different variables & slots

```python
list2 = myList
circ = list2[5]
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```

```
myList
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
list2
[98.6, 17, 23, True, False, 'I' 'was' 'not' 'Adams']
```

```
Circle
radius 50
location
x 200
y 100
```

```
Point
x 200
y 100
```
Aliasing: the very same object can be stored in different variables & slots

```python
list2 = myList
circ = list2[5]
myList[1][3] = circ
```

```
myList
0 1 2 3 4 5 0 1 2 3
'98.6' '23' 'True' '0' '100' 'radio' '50' 'location' '200' '100'

I was 'not' Adams
```

```
circ
0 1 2 3 4 5
0 2 3

Point
x: 200
y: 100
```

```
myList
0 1 2 3 4 5 0 1 2 3
'98.6' '23' 'True' '0' '100' 'radio' '50' 'location' '200' '100'

I was 'not'
```

```
circ
0 1 2 3 4 5
0 2 3

Point
x: 200
y: 100
```

Aliasing: the very same object can be stored in different variables & slots

```python
list2 = myList
circ = list2[5]
myList[1][3] = circ
```

```
myList
0 1 2 3 4 5 0 1 2 3
'98.6' '23' 'True' '0' '100' 'radio' '50' 'location' '200' '100'

I was 'not'
```

```
circ
0 1 2 3 4 5
0 2 3

Point
x: 200
y: 100
```

```
myList
0 1 2 3 4 5 0 1 2 3
'98.6' '23' 'True' '0' '100' 'radio' '50' 'location' '200' '100'

was a
```

```
circ
0 1 2 3 4 5
0 2 3

Point
x: 200
y: 100
```

```
myList
0 1 2 3 4 5 0 1 2 3
'98.6' '23' 'True' '0' '100' 'radio' '50' 'location' '200' '100'

was a
```

```
circ
0 1 2 3 4 5
0 2 3

Point
x: 200
y: 100
```
Aliasing: the very same object can be stored in different variables & slots

```python
list2 = myList
circ = list2[5]
myList[1][3] = circ
```

```text
myList[1][2] = 'a'
list2[4][2] = 'a'
```

List Diagrams/Mutability 8-37

```python
list2 = myList
circ = list2[5]
myList[1][3] = circ
```

```text
myList[1][2] = 'a'
list2[4][2] = 'a'
```

List Diagrams/Mutability 8-38

Aliasing: the very same object can be stored in different variables & slots

```python
list2 = myList
circ = list2[5]
myList[1][3] = circ
```

```text
myList[1][2] = 'a'
list2[4][2] = 'a'
```

List Diagrams/Mutability 8-39

```python
list2 = myList
circ = list2[5]
myList[1][3] = circ
```

```text
myList[1][2] = 'a'
list2[4][2] = 'a'
```

List Diagrams/Mutability 8-40
Aliasing: the very same object can be stored in different variables & slots

```python
list2 = myList
circ = list2[5]
myList[1][3] = circ

list2 = myList
circ = list2[5]
myList[1][3] = circ
```

List Diagrams/Mutability 8-41
What is the final value of \( c[0] \)?

\[
a = [15, 20] \\
b = [15, 20] \\
c = [10, a, b] \\
b[1] = 2*a[0] \\
c[1][0] = c[0] \\
c[0] = a[0] + c[1][1] + b[0] + c[2][1]
\]

Draw a memory diagram!

Does the answer change if we change the 2nd line from

\[
b = [15, 20] \quad \text{to} \quad b = a[:]
\]

Does the answer change if we change the 2nd line from

\[
b = [15, 20] \quad \text{to} \quad b = a
\]

Lists are mutable. What about strings?

Strings are sequences:

\[
\begin{align*}
\text{In [6]}: & \quad \text{name} = 'Gryffindor' \\
\text{In [7]}: & \quad \text{name[2]} \quad \# 'y' \\
\text{In [8]}: & \quad \text{name[4:8]} \quad \# 'find' \\
\text{In [9]}: & \quad \text{'do' in name} \quad \# True
\end{align*}
\]

Mutation operations do not work on strings:

\[
\begin{align*}
\text{In [10]}: & \quad \text{name[4]} = 't' \quad \# \text{what happens?} \\
\text{In [11]}: & \quad \text{name.append('s')} \quad \# \text{what happens?}
\end{align*}
\]

Strings are immutable sequences.

Once you create a string, it cannot be changed

\[
\begin{align*}
\text{In[13]}: & \quad \text{college} = 'WELLESLEY' \\
\text{In[14]}: & \quad \text{college.lower( )} \\
\text{Out[14]}: & \quad 'wellesley' \quad \# \text{Returns a new string 'wellesley';} \\
& \quad \# \text{old one is unchanged!}
\end{align*}
\]

\[
\begin{align*}
\text{In[15]}: & \quad \text{myCollege = college.lower( )} \\
\text{myCollege} & \quad 'wellesley'
\end{align*}
\]

Tuples

Lists are mutable sequences of values.

Tuples are immutable sequences of values.

Tuples are written as comma-separated values delimited by parentheses.

\[
\begin{align*}
\# \text{A homogeneous tuple of five integers} & \quad (5, 8, 7, 1, 3) \\
\# \text{A homogeneous tuple of four strings} & \quad ('Gryffindor', 'Hufflepuff', 'Ravenclaw', 'Slytherin') \\
\# \text{A heterogeneous tuple of three elements} & \quad (42, 'Hello', False) \\
\# \text{A pair is a tuple with two elements} & \quad (7, 3) \\
\# \text{A tuple with one element must use a comma to avoid} & \quad (7, ) \\
\# \text{being confused with a parenthesized expression} & \quad () \\
& \quad \# \text{A tuple with 0 values}
\end{align*}
\]
Tuples are immutable sequences.

Like strings, tuples support all sequence operations that do not involve mutation.

```python
In[32]: houseTuple = ('Gryffindor', 'Hufflepuff', 'Ravenclaw', 'Slytherin')
Out[32]:
In[33]: houseTuple[2]
Out[33]: 'Gryffindor'
In[34]: houseTuple[1:3]
Out[34]: ('Hufflepuff', 'Ravenclaw')
In[35]: houseTuple.count('Slytherin')
Out[35]: 1
In[36]: 'Ravenclaw' in houseTuple
Out[36]: True
In[37]: houseTuple * 2 + ('12 Grimmauld Place',)
Out[37]: ('Gryffindor', 'Hufflepuff', 'Ravenclaw', 'Slytherin', 'Gryffindor', 'Hufflepuff', 'Ravenclaw', 'Slytherin', '12 Grimmauld Place')
```

Mutation operations do not work on tuples.

```python
In [38]: houseTuple[0] = '4 Privet Drive'
----------------------------------------------------------
TypeError...
houseTuple[0] = '4 Privet Drive'
TypeError: 'tuple' object does not support item assignment
In [39]: houseTuple.append('The Shrieking Shack')
-----------------------------------------------------------
AttributeError...
houseTuple.append('The Shrieking Shack')
AttributeError: 'tuple' object has no attribute 'append'
In [40]: houseTuple.pop(1)
-----------------------------------------------------------
AttributeError...
houseTuple.pop(1)
AttributeError: 'tuple' object has no attribute 'pop'
```

Conversion between sequence types

The built-in functions `str`, `list`, `tuple` create a new value of the corresponding type.

```python
In [41]: word = "Wellesley"
In [42]: list(word)
Out[42]: ['W', 'e', 'l', 'l', 'e', 's', 'l', 'e', 'y']
In [43]: tuple(word)
Out[43]: ('W', 'e', 'l', 'l', 'e', 's', 'l', 'e', 'y')
In [44]: numbers = range(5, 15, 2)
In [45]: str(numbers)
Out[45]: '5, 7, 9, 11, 13'
```

Enumerations

When called on a sequence, the `enumerate` function returns a sequence of pairs of indices and values.

```python
In [46]: list(enumerate('boston'))
Out[46]: [(0, 'b'), (1, 'o'), (2, 's'), (3, 't'), (4, 'o'), (5, 'n')]
In [47]: list(enumerate([7, 2, 8, 5]))
Out[47]: [(0, 7), (1, 2), (2, 8), (3, 5)]
In [48]: for (index, char) in enumerate('boston'):
    ...:     print(index, char)
0 b
1 o
2 s
3 t
4 o
5 n
```