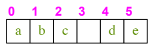
Lists: Sequences of Objects

String

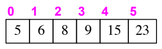
- s = 'abc de'



- Put characters in quotes
  - Use \' for quote character
- Access characters with [ ]
  - s[0] is 'a'
  - s[6] causes an error
  - s[0:2] is 'ab' (excludes c)
  - s[2:] is 'c de'

List

- x = [5, 6, 5, 9, 15, 23]



- Put items inside [ ]
  - Separate by commas
- Access items with [ ]
  - x[0] is 5
  - x[6] causes an error
  - x[0:2] is [5, 6] (excludes 2nd 5)
  - x[3:] is [9, 15, 23]

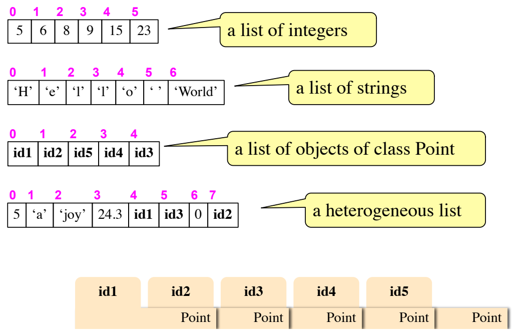
Things that Work for All Sequences

s = 'slithy'		x = [5, 6, 9, 6, 15, 5]
s.index('s') → 0	methods	x.index(5) → 0
s.count('t') → 1	methods	x.count(6) → 2
len(s) → 6	built-in fn.	len(x) → 6
s[4] → "h"	slicing	x[4] → 15
s[1:3] → "li"	slicing	x[1:3] → [6, 9]
s[3:] → "thy"	slicing	x[3:] → [6, 15, 5]
s[–2] → "h"	slicing	x[–2] → 15
s + ' toves' → "slithy toves"	operators	x + [1, 2] → [5, 6, 9, 6, 15, 5, 1, 2]
s * 2 → "slithyslithy"	operators	x * 2 → [5, 6, 9, 6, 15, 5, 5, 6, 9, 6, 15, 5]

the smallest *i* for which x[i] == 5

the number of *i*s for which x[i] == 6

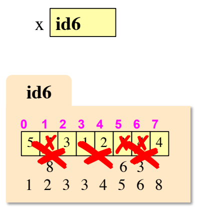
Difference: Lists Hold Any Type



Difference: Lists are mutable

- Their contents **can be altered**
  - by assignment to list items
    x = [5, 7, 3, 1]
    x[1] = 8
  - using methods
    x.append(2)
    x.extend([3, 4])
    x.insert(5, 6)
    x.sort()
- Draw lists as folders
  - because they are mutable objects
  - can omit type to save space

See Python Standard Library for more methods

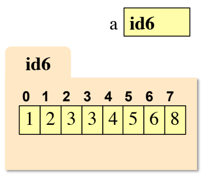


Does not work for strings
s = 'Hello World!'
s[0] = 'J' **ERROR**
s.append('?') **ERROR**

Lists vs. Objects With Attributes

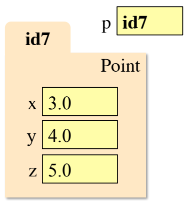
List

- Attributes are indexed
  - Example: a[2]



Point

- Attributes are named
  - Example: p.x



Clicker Exercise

- Execute the following:
```
>>> x = [5, 6, 5, 9, 10]
>>> x[3] = -1
>>> x.insert(1, 2)
```
- What is x[4]?

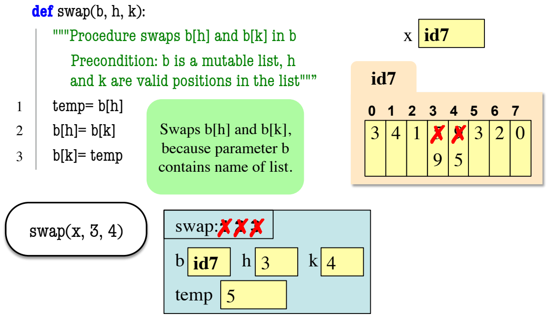
A: 10
B: 9
C: -1
D: **ERROR**
E: I don't know

- Execute the following:
```
>>> x = [5, 6, 5, 9, 10]
>>> y = x
>>> y[1] = 7
```
- What is x[1]?

A: 7
B: 5
C: 6
D: **ERROR**
E: I don't know

Lists and Functions: Swap

```
def swap(b, h, k):
    """Procedure swaps b[h] and b[k] in b
      Precondition: b is a mutable list, h
      and k are valid positions in the list"""
    temp= b[h]
    b[h]= b[k]
    b[k]= temp
```

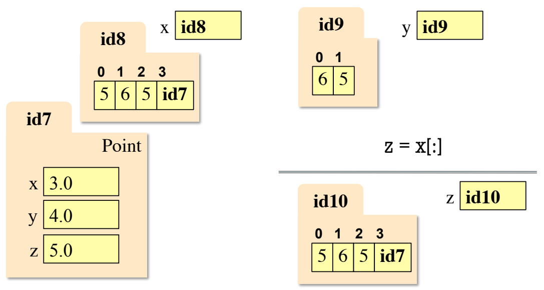


Slicing Lists Makes Copies

x = [5, 6, 5, Point(3,4,5)]

y = x[1:3]

z = x[:]



Clicker Exercise

- Execute the following:
  >>> x = [5, 6, 5, 9, 10]
  >>> y = x[1:]
  >>> y[0] = 7
- What is x[1]?

A: 7
B: 5
C: 6
D: **ERROR**
E: I don't know

- Execute the following:
  >>> x = [5, Point(1, 2, 3), 6]
  >>> y = x[1:]
  >>> y[0].x = 7
- What is x[1].x?

A: 1
B: 5
C: 7
D: **ERROR**
E: I don't know

Lists and Strings: They go together like…

text.split(sep): return a list of the words in text (separated by sep, or whitespace by default)

sep.join(words): concatenate the items in the list of strings words, separated by sep.

```
text = 'Rama lama lama\nke ding a de ding a dong'
words = text.split()          ['Rama', 'lama', 'lama', 'ke', …]
lines = text.split('\n')      returns a list of two strings
sep = '-'
print sep.join(words)         'Rama-lama-lama-ke…'
s = (sep.join(lines[0].split()) + ' ' + sep.join(lines[1].split())
                              'Rama-lama-lama ke-ding-a-de-ding-a-dong'
```

…a horse and carriage? Bread and butter?

Tuples

strings: **immutable** sequences of **characters**

lists: mutable sequences of **any objects**

tuples: **immutable** sequences of **any objects**

"tuple" generalizes "pair," "triple," "quadruple," …

- Tuples fall between strings and lists
  - write them with just commas: 42, 4.0, 'x'
  - often enclosed in parentheses: (42, 4.0, 'x')

length 1: (42,)
length 0: ()

Conventionally use lists for:
- long sequences
- homogeneous sequences
- variable length sequences

Conventionally use tuples for:
- short sequences
- heterogeneous sequences
- fixed length sequences