Middlebury

CSCI 146: Intensive Introduction to Computing

Fall 2025

Lecture 10: Sets and Dictionaries

Goals for today

- Explain when sets are used
- Describe sets as unordered unique collections of objects of arbitrary type
- Create a set
- Explain and use functions, methods and operators on sets including adding, deleting, membership, etc.
- Explain the properties of different data structures
- Explain when dictionaries are used
- Describe dictionaries as a key-value store
- Create a dictionary
- Explain and use functions, methods and operators on dictionaries including adding, indexing, deleting, etc.

Change in coursework: there will be ONE Test Project released in Week 11 (not two).

Motivation: imagine you had a `list` called `mylist` and wanted to know if `x` was in `mylist`.

```
def contains(mylist, x):
  for item in mylist:
    if item == x:
      return True
  return False
```

How long would this take if `mylist` had 100 items and each `item == x` check took 1 second?

Introducing sets: a data structure to store *unordered*, *unique* items.

```
>>> help(set)
>>> s = set() # uses the set "initializer" to create an empty set
>>> s = {1, 2, 3, 4} # delimited with curly braces
>>> s.add(5)
>>> s.add(2)
>>> set("abcd") # uses initializer that accepts "iterable"
>>> set("abcda")
```

Other methods (some mutate the `set`, some don't):

- `add`: Mutate
- `clear` : Mutate
- `union`: Non-mutating (can also use `|` operator)
- `update`: Mutating (update is effectively a union operation)
- `intersection` : Non-mutating (can also use `&` operator)
- `intersection_update` : Mutate
- `difference` : Non-mutating (can also use `-` operator)
- `difference_update` : Mutate
- `symmetric_difference`: Non-mutating (can also use `^` operator).

Membership can be checked with `in` (as with `list`s and `str`ings).

A mini note about `set()`.

- This uses the "initializer" (constructor in other programming languages) which creates an *instance* of the `set` *class*.

Examples with `set` operators.

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket) # duplicates have been removed
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket   # fast membership testing
True
>>> 'crabgrass' in basket
False
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>>  a - b  # letters in a but not in b
{'r', 'd', 'b'}
>>> a | b # letters in a or b or both
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b # letters in both a and b
{'a', 'c'}
>>> a ^ b # letters in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

Question 1: Which of the following is equivalent to the value of the set `s`?

```
>>> s = set("abcabc")
```

A. {"abcabc"}

B. {"abc","abc"}

C. {"a","b","c","a","b","c"}

D. {"a","b","c"}

E. {"abc"}

Question 2: Which of the following produces the same value for c (including type) as the code below?

```
a = [1, 2, 3]
b = [3, 4, 5]
c = []
for val in a:
    if val in b:
        c.append(val)
```

A. `c = set(a + b)`

B. `c = list(set(a) & set(b))`

C. `c = list(set(a) | set(b))`

D. `c = set(a) & set(b)`

E. `c = list(a & b)`

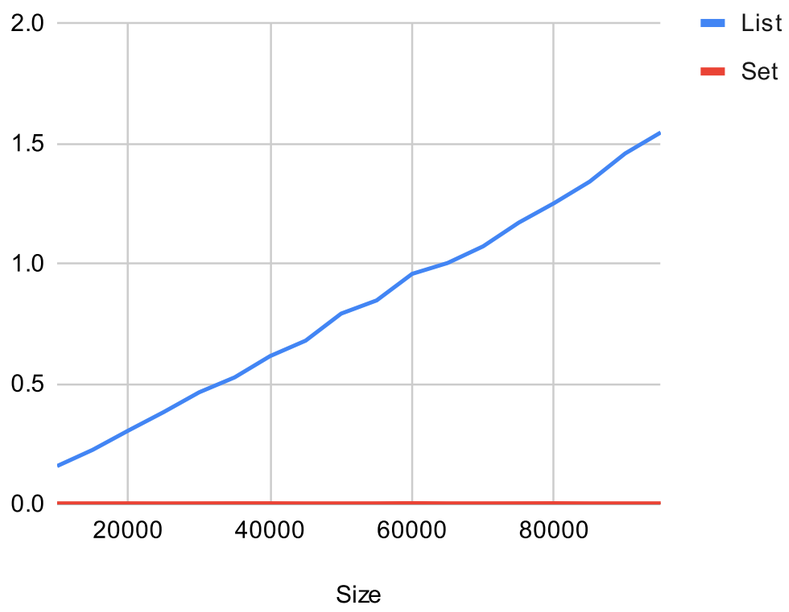
{3} → [3]

Why not a `list`? Performance for operators like "contains" (membership).

- But we lose *ordering* (maybe that's okay depending on the application).
- Maybe we want the uniqueness property.
- Can't "index" into a set, but we can still iterate.

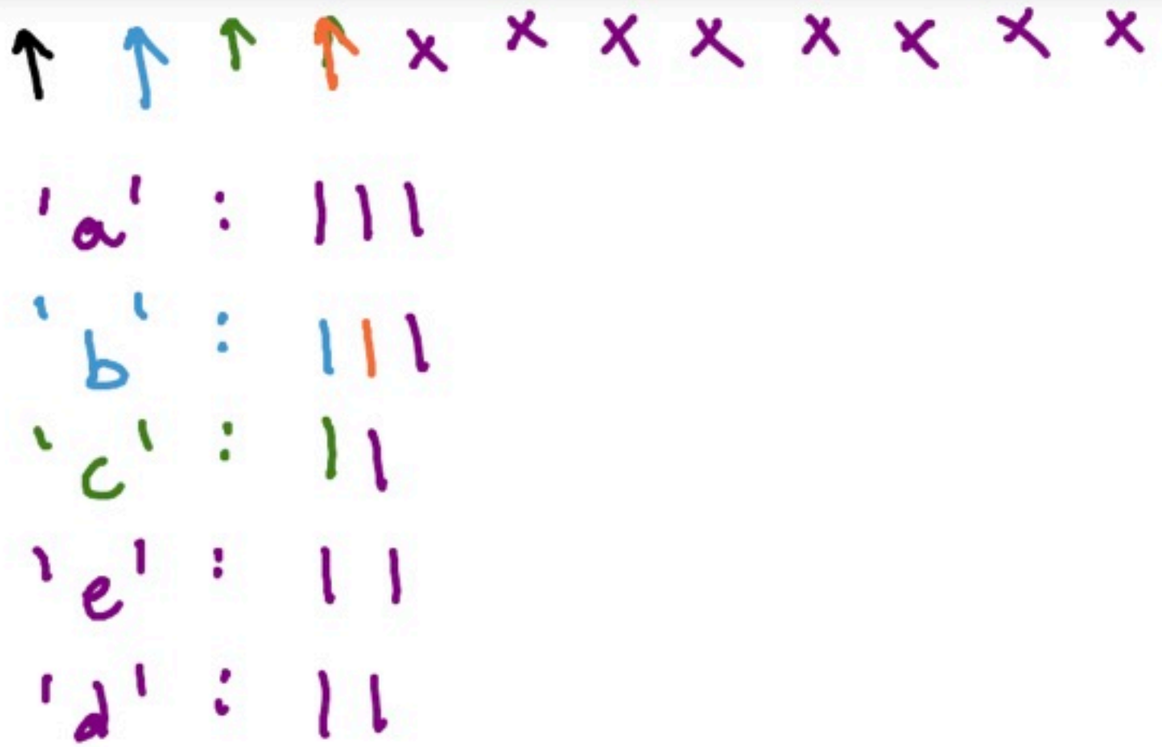
```
s = {"a", "b", "c", "d"}
for item in s:
  print(s)
```

Run `list_versus_set.py`.



Exercise: determine the frequency of each letter below.

```
['a', 'b', 'c', 'b', 'c', 'b', 'a', 'a', 'e', 'd', 'd', 'e']
```



Dictionaries allow us to store *key-value pairs*.

- Also known as "maps" or "associative arrays".
- Dictionary literal delimited by curly-braces.
- Can initialize dictionary (but not `set`) with `{}`.
- Can also initialize empty dictionary with `dict()`.

```
>>> d = { 5: 1, 6: 2 }
>>> d
{5: 1, 6: 2}
>>> d[5]
1
>>> d[6]
2
>>> d[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 1
>>> d.get(1, 6)
6
>>> d.get(5, 6)
1
>>> d[3] = 7
>>> d
{3: 7, 5: 1, 6: 2}
```

Question 3: Which of the following is best suited for a dictionary instead of a list?

A. The order in which people finish a race.

B. The ingredients necessary for a recipe.

C. The names of world countries and their capital cities.

D. 50 random integers.

Question 4: What is the dictionary created by the code below?

```
>>> d = {3:4}
>>> d[5] = d.get(4, 8)
>>> d[4] = d.get(3, 9)
```

A. `{3:4, 5:8, 4:9}`

B. `{3:4, 5:8, 4:4}`

C. `{3:4, 5:4, 4:3}`

D. This code has an error.

Question 5: What is the dictionary created by the code below?

```
>>> d = {1:5}
>>> d[2] = d[1]
>>> d[4] = d[3]
```

A. `{1:5, 2:5}`

B. `{1:5, 2:1}`

C. `{1:5, 2:5, 4:None}`

D. This code has an error.

Iterating through dictionaries.

- Use `.keys()` to retrieve keys.
- Use `.items()` to retrieve key-value pairs.

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

for k in d.keys():
  print(k)
```

```
for item in d.items():
  print(item)

for k, v in d.items():
  print(k, v)
```

Summary and Reminders

- Programming Assignment 5 due Thursday (can work and submit in pairs).
- Tuples next time.

Type	Ordered	Mutable	Mutable Values	Typical (but not only) Usage
List	Yes	Yes	Yes	Ordered collection of variable length (often homogenous)
Set	No	Yes	No	Membership/Set operations
Tuple	Yes	No	Yes	Heterogeneous (ordered) collection of fixed length
Dictionary	Yes-ish	Yes	Yes (but not keys)	Key -> Value lookup