CS134:

Dictionaries and Sets

Announcements & Logistics

- **Practice midterm** on Glow under Files
- **Lab 5** due Friday at noon for everyone
  - We're still working on grading Lab 4
- **Midterm tomorrow in TCL 123**:
  - Thu Oct 20 6 - 7:30pm, 8 - 9:30pm
  - TCL 206 reserved for reduced distractions/extra time (pick up exam from Jeannie/Iris in TCL 123)
  - Closed books and notes
- **No class** Fri Oct 21st!
  - Lab 6 will be released Friday
  - Read over before Mon/Tue

Do You Have Any Questions?

Last Time

- A **dictionary** is a **mutable** collection that maps **keys** to **values**
    - **Keys** must be unique & **immutable, values** can any Python object
- Iterating over a dictionary: what do we iterate over?
    - Iterate over the ***keys*** of a dictionary directly (by default)
- Dictionary comprehensions: similar to list comprehensions
- Learned about useful dictionary methods:
    - `dict.get(key, defaultVal)`
    - `dict.values(), dict.items(), dict.keys()`

Recap: Dictionary `.get()` method

- `dict.get(key, defaultVal)`
  - If key exists, `dict.get(key, defaultVal)` returns the value, just like `dict[key]`
  - If the key does not exist, `dict.get(key, defaultVal)` returns `defaultVal`
  - If the key does not exist, `dict[key]` always returns a `KeyError`
  - `defaultVal` is optional
- `dict.get(key)`
  - If key exists, `dict.get(key)` returns the value, just like `dict[key]`
  - If the key does not exist, `dict.get(key)` returns `None`

Recap: Iterating Over a Dictionary

```
calendar = {"Jan": 31, "Feb": 28, "Mar": 31, "Apr": 30,
            "May": 31, "Jun": 30, "Jul": 31, "Aug": 31,
            "Sep": 30, "Oct": 31, "Nov": 30, "Dec": 31}
```

- We **iterate over the keys** of a dictionary directly in a for loop

```
>>> for month in calendar:
...     print(month, end=" ")
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

- To **iterate over values** we can use the `.values()` method

```
 >>> for days in calendar.values():
    ...     print(days, end=" ")
31 28 31 30 31 30 31 31 30 31 30 31
```

- To **iterate over key, value pairs** we can use the `.items()` method

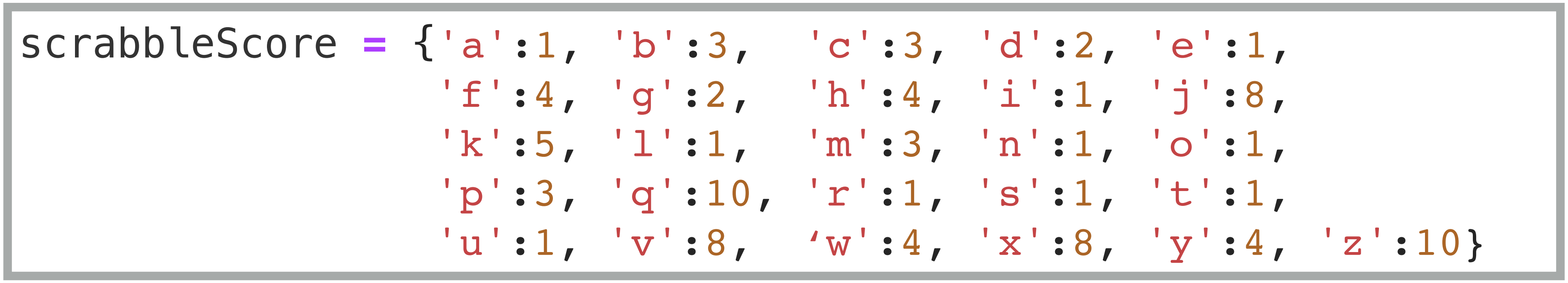
```
>>> for month, days in calendar.items():
>>> ...     print(month, days, end=" ")
Jan 31 Feb 28 Mar 31 Apr 30 May 31 Jun 30 Jul 31
Aug 31 Sep 30 Oct 31 Nov 30 Dec 31
```

Today's Plan

- Wrap up dictionaries
- Investigate **sorting** with dictionaries
- (Briefly) Discuss another unordered data structure: **sets**
- Review all data structures so far and when to use each

Sorting Operations with Dictionaries

- Let's say we're developing a Scrabble app
- We can store the score for each letter as a dictionary as below



```
scrabbleScore = {'a':1, 'b':3,  'c':3, 'd':2, 'e':1,
                 'f':4, 'g':2,  'h':4, 'i':1, 'j':8,
                 'k':5, 'l':1,  'm':3, 'n':1, 'o':1,
                 'p':3, 'q':10, 'r':1, 's':1, 't':1,
                 'u':1, 'v':8,  ‘w':4, 'x':8, 'y':4, 'z':10}
```

- If we call the `sorted()` function on a dictionary, it returns an **ordered list of all the keys**.

```
>>> print(sorted(scrabbleScore))
```

Sorting Operations with Dictionaries

- Let's say we're developing a Scrabble app
- We can store the score for each letter as a dictionary as below

```
scrabbleScore = {'a':1, 'b':3,  'c':3, 'd':2, 'e':1,
                 'f':4, 'g':2,  'h':4, 'i':1, 'j':8,
                 'k':5, 'l':1,  'm':3, 'n':1, 'o':1,
                 'p':3, 'q':10, 'r':1, 's':1, 't':1,
                 'u':1, 'v':8,  ‘w':4, 'x':8, 'y':4, 'z':10}
```

- If we call the `sorted()` function on a dictionary, it returns an **ordered list of all the keys**.

```
>>> print(sorted(scrabbleScore))
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x',
'y', 'z']
```

Sorting By Value

- This behavior isn't super useful for Scrabble
- What might we want instead?

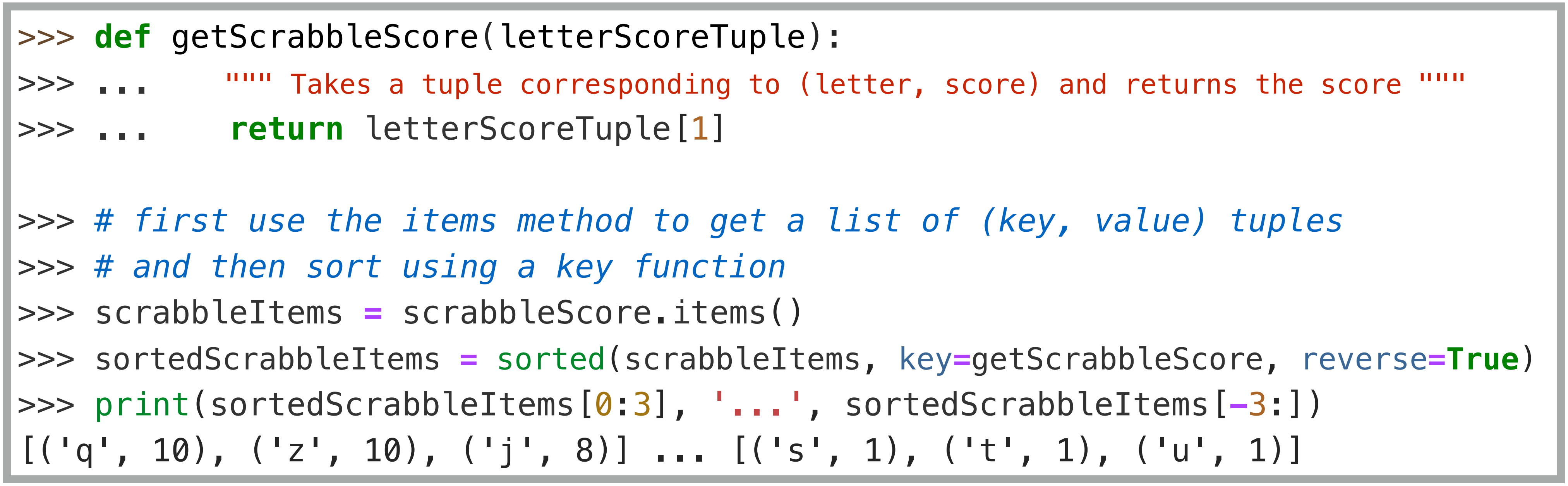
Sorting By Value

- This behavior isn't super useful for Scrabble
- What might we want instead?
  - Sort based on the scores of the letters (from highest to lowest)
- This known as a **sort-by-value** as opposed to **sort-by-key**
- As before, using `sorted()` with a `key` function (not be confused with the keys in the dictionary! 😬) comes in handy.
- We'll need to spend just a little more effort to come up with a suitable `key` function for dictionaries
- Ex: Jupyter notebook

Sorting By Value

- We first use the **`items()`** method to generate a list of tuples, where each tuple is a key-value pair

- We then sort this list based on value (second element of each tuple.)



```
>>> def getScrabbleScore(letterScoreTuple):
>>> ...    """ Takes a tuple corresponding to (letter, score) and returns the score """
>>> ...    return letterScoreTuple[1]

>>> # first use the items method to get a list of (key, value) tuples
>>> # and then sort using a key function
>>> scrabbleItems = scrabbleScore.items()
>>> sortedScrabbleItems = sorted(scrabbleItems, key=getScrabbleScore, reverse=True)
>>> print(sortedScrabbleItems[0:3], '...', sortedScrabbleItems[-3:])
[('q', 10), ('z', 10), ('j', 8)] ... [('s', 1), ('t', 1), ('u', 1)]
```

- We can also use a list comprehension after to extract just the keys if desired.

Sets

New Unordered Data Structure: Sets

- Dictionaries are collections of unordered **key, value** pairs
- What if we only need an unordered collection of individual items?
    - We can use a new data structure: **sets**
- Sets are ***mutable***, **unordered** collections of **immutable** objects
- Sets are written as comma separated values between curly braces
- Like keys in a dictionary, values in a set must be **unique** and **immutable**
    - Sets can be an effective way of **eliminating duplicate values**

```
>>> nums = {42, 17, 8, 57, 23}
>>> flowers = {"tulips", "daffodils", "asters", "daisies"}
>>> people = {("Charlie", "Brown"), ("Lucy", "Van Pelt"),
              ("Franklin", "Armstrong")}
>>> emptySet = set() # empty set
```

New Unordered Data Structure: Sets

- **Question:** What is the potential downside of removing duplicates w/sets?

```
>>> firstChoice = {'a', 'b', 'a', 'a', 'b', 'c'}
>>> uniques = set(firstChoice)
>>> uniques
# ???
>>> set("aabrakadabra")
# ???
```

New Unordered Data Structure: Sets

- **Question:** What is the potential downside of removing duplicates w/sets?
  - Might lose the ordering of elements

```
>>> firstChoice = {'a', 'b', 'a', 'a', 'b', 'c'}
>>> uniques = set(firstChoice)
>>> uniques
{'a', 'b', 'c'}
>>> set("aabrakadabra")
{'a', 'b', 'd', 'k', 'r'}
```

Sets: Membership and Iteration

- Can check membership in a **set** using `in`, `not in`
- Can check length of a set using `len()`
- Can iterate over values in a loop (order will be arbitrary)

```
>>> nums = {42, 17, 8, 57, 23}
>>> flowers = {"tulips", "daffodils", "asters", "daisies"}
>>> 16 in nums
False
>>> "asters" in flowers
True
>>> len(flowers)
4
>>> # iterable
>>> for f in flowers:
>>> ...   print(f, end=" ")
tulips daisies daffodils asters
```

end = " " prevents new line

Sets are Unordered

- Therefore we **cannot**:
  - Index into a set (no notion of "position")
  - Concatenate two sets (concatenation implies ordering)
  - Create a set of ***mutable*** objects:
    - Such as lists, sets, and dictionaries

```
>>> {[3, 2], [1, 5, 4]}
TypeError
----> 1 {[3, 2], [1, 5, 4]}

TypeError: unhashable type: 'list'
```

Set Methods Summary

- We can use set methods to manipulate sets
- `s.add(item)`: changes the set `s` by adding `item` to it
- `s.remove(item)`: changes the set `s` by removing `item` from `s`.
    - If item is not in `s`, a `KeyError` occurs

The following operations always return a new set.

- `s1.union(s2)` or `s1 | s2`: returns a new set that has all elements that are either in `s1` or `s2`
- `s1.intersection(s2)` or `s1 & s2`: returns a new set that has all the elements that are in both sets.
- `s1.difference(s2)` or `s1 - s2`: returns a new set that has all the elements of `s1` that are not in `s2`
- `s1 |= s2`, `s1 &= s2`, `s1 -= s2` are versions of `|, &, -` that mutate `s1` to become the result of the operation on the two sets.

An Overview of Python Data Structures (so far!)

Python Data Structures at a Glance

	Lists	Tuples	Dictionaries	Sets
Order	Yes	Yes	No	No
Mutability	Yes	No	Yes (keys are immutable)	Yes (items are immutable)
Iterable	Yes	Yes	Yes	Yes
Comprehensions	Yes	Yes (need to enclose in `tuple`)	Yes	Yes

Methods	`.append(),`
	`.extend(),`
	`.count(),`
	`.index(),`
	`etc`
		`.count(),`
		`.index(),`
			`.get(),`
			`.pop(),`
			`etc`
				`.add(),`
				`.remove(),`
				`etc`

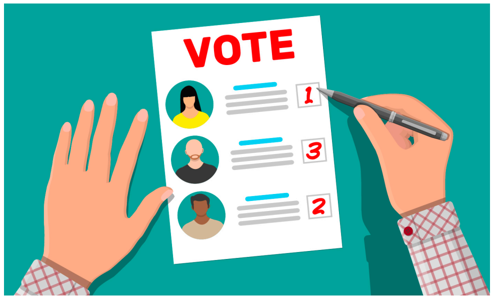
Python Data Structures at a Glance

	Lists	Tuples	Dictionaries	Sets
Order	Yes	Yes	No	No
Mutability	Yes	No	Yes (keys are immutable)	Yes (items are immutable)
Iterable	Yes	Yes	Yes	Yes
Comprehensions	Yes	Yes (need to enclose in `tuple`)	Yes	Yes

	`d(),`
	`d(),`
	`(),`
		`.count(),`
		`.index(),`
			`.get(),`
			`.pop(),`
				`.add(),`
				`.remove(),`
				`etc`

Which to use when?

Does Order Matter?



- Examples where **order** in data is important:
    - *Ranked* ballots
    - Queues
    - Words in a sentence
    - Tables/Matrices

- Tuples or lists?
    - Do we need to **add/remove items dynamically**?
        - If yes, use **lists** (they are mutable!)
    - If data stays same (no changes), use **tuples** (more space efficient)
    - Even though you can concatenate items to tuples, it is not efficient, as it requires "copying over all the data" and creating a new tuple

Unordered Collections

- When storing a collection of data with ***no implicit ordering***:
  - Use **dictionaries** or **sets**
  - Dictionaries are more appropriate when there is a ***key, value pair***
  - Better performance in general as compared to ordered structures
- Convenient when we want to store data with different attributes and support quick attribute lookups
- Can store a **dictionary of dictionaries** (just like lists of lists!)

```
peanutsDict = { 'cb23': {'name': 'Charlie Brown',
                         'age':8,
                         'icecream': 'cookie dough'},

                 'pp3': {'name': 'Peppermint Patty',
                         'age': 7,
                         ‘icecream': 'peppermint'},

                 'sd4': {'name': 'Snoopy Dog',
                         'age': 72,
                         ‘icecream': 'vanilla'}}
```

The end!