Lecture 10:
Dictionaries and Sets
Check-in and Reminders

• Submit **Homework 3** in the box up front

• Remember that this week’s lab is partnered
  • Partner must be in the same lab section
  • If you have not found a partner yet, check out the shared google doc to find students who are also looking

• Heads up. Midterm is **Thursday, March 12**
  • **Closed book exam**
  • Review homework and lectures: best practice for exams
  • Exact Syllabus will be announced Wed

*Do You Have Any Questions?*
Sequence vs Collection

- **Sequence:** a group of items that come one after the other (there is an ordering of items)

- **Collection:** a group of things brought together for some purpose

- Is a sequence a collection? Is a collection a sequence?
Python Collections

- A sequence is an ordered collection in which elements can be accessed by their index.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>a mutable sequence of arbitrary objects</td>
<td>[-100, &quot;blue&quot;, (1, 10), True]</td>
</tr>
<tr>
<td>tuple</td>
<td>an immutable sequence of arbitrary objects</td>
<td>(2017, &quot;Mar&quot;, 2)</td>
</tr>
<tr>
<td>string</td>
<td>an immutable sequence of characters</td>
<td>&quot;Go Wellesley!&quot;</td>
</tr>
<tr>
<td>range</td>
<td>an immutable sequence of numbers</td>
<td>range(3)</td>
</tr>
<tr>
<td>set</td>
<td>a mutable unordered collection of distinct objects.</td>
<td>{1, 4, 5, 23}</td>
</tr>
<tr>
<td>dict</td>
<td>a mutable unordered collection of key:value pairs, where keys are immutable and values are any Python objects</td>
<td>{&quot;orange&quot;: &quot;fruit&quot;, 3: &quot;March&quot;, &quot;even&quot;: [2, 4, 6, 8]}</td>
</tr>
</tbody>
</table>
Properties of Sequences & Collections

• Collections (list, tuple, string, range, set, dict)
  • Find their length with \texttt{len()}
  • Check element membership in the collection with \texttt{in}
  • Are iterables (we can iterate over their elements in a loop)

• Sequences (list, tuple, string, range)
  • Use indices to access elements, e.g., \texttt{myList[2]}
  • Use slice operations for subsequences, e.g., \texttt{myList[2:4]}

• Mutable (list, set, dict): can be changed through object methods
• Immutable (tuple, string, range): cannot be changed
A New Mutable Collection: Sets

• Sets are written as comma separated values between curly braces

nums = {42, 17, 8, 57, 23}
animals = {'duck', 'cat', 'bunny', 'ant'}
potters = {('Ron', 'Weasley'), ('Luna', 'Lovegood'), ('Harry', 'Potter')}
vowels = {}  # empty set

• The values in a set must be immutable objects, just like keys in a dictionary.
• So sets cannot include as values lists, dictionaries, or even sets
Properties of Sets

- **Elements of sets are unordered.** When a set is printed, the order of elements in unpredictable.
  - Jupyter notebooks, however, show returned set elements in ascending order even though they are fundamentally unordered.

- **Sets contain no duplicates.** An element is either contained in a set or not. An element cannot appear more than once in a set.
  - Sets are thus an effective way of removing duplicates!

```
In [11] listWithDups = [4, 1, 3, 2, 3, 4, 1]
In [12] set(listWithDups)
Out [12] {1, 2, 3, 4}
In [13] list(set(listWithDups))
Out [13] [1, 2, 3, 4]
```
Dictionaries

- A **dictionary** is a **mutable** collection that maps **keys** to **values**
- A dictionary is enclosed with curly brackets and contains comma-separated pairs. A pair is a colon-separated key and value.

```python
daysOfMonth = {'Jan' : 31, 'Feb' : 28, 'Mar' : 31,... }  
```

```python
monthsLength = {
30: ['Apr', 'Jun', 'Sep', 'Nov'],
28: [Feb] }
```

- **Keys**: an **immutable** type such as numbers, strings, or tuples
- **Values**: any Python object (numbers, strings, lists, tuples, etc.)
Creating Dictionaries

• Direct assignment: provide keys, value pairs delimited with `{}`

In [1] scrabbleDict = {'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': 4, 'w': 4, 'x': 8, 'y': 4, 'z': 10}

• Start with empty dict and add key, value pairs

In [2] cart = {}  # an empty dict
In [4] cart['kiwis'] = 2.54
In [5] cart

• Applying the built-in constructor function `dict` to a list of tuples:

Out [6] {'Harry': 12, ‘Hagrid’: 40}

Note. keys may be listed in any order
Dictionary Operations

• The value associated with a key is accessed using the same subscripting notation with square brackets used for list indexing

```
In [1] daysOfMonth = {'Jan' : 31, 'Feb' : 28, 'Mar' : 31, ... }

In [2] daysOfMonth['Oct']
Out [2] 31

In [3] scrabbleDict['z']
Out [3] 10
```
Before accessing a dictionary with a key, you should check if the key exists using the `in` operation.

```
In [4] daysOfMonth['October']
---
KeyError Traceback (most recent call last)
  in ()
  1 daysOfMonth['October']
KeyError: 'October'

In [5] 'Oct' in daysOfMonth
Out [5] True

In [6] 'October' in daysOfMonth
```
Dictionaries and Mutability

**Dictionaries are mutable**

- We can add or remove key-value pairs
- We can change the value of an existing key

In [7] days0fMonth['Feb'] = 29

**Dictionary Keys are Immutable**

- Example, a list or dict cannot be a key of a dictionary (only immutable objects such as numbers, strings, tuples) can be keys
Dictionary Methods: get

• The **get** method is an alternative to using subscript to get the value associated with a key in a dictionary. It takes two arguments:
  • the key
  • an optional default value to use if the key is not in the dictionary

In [8] daysOfMonth.get('Oct', 'unknown')
Out [8] 31

In [9] daysOfMonth.get('OCT', 'unknown')
Out [9] 'unknown'

• If the optional second argument is omitted and the key does not exists, get will return **None**.

In [10] print(daysOfMonth.get('OCT'))
Out [10] None
Dictionary Methods: keys, values, items

- Sometimes we are interested in knowing the **keys**, **values** or **items** (key, value pairs) of a dictionary.
- Each of these methods returns an object containing only the keys, values, and items, respectively.

```python
In [164]: daysOfMonth.keys()
```

```python
In [166]: daysOfMonth.values()
Out[166]: dict_values([31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31])
```

```python
In [168]: daysOfMonth.items()
```
Iterating over/membership in Dicts

When iterating over the keys in a dictionary, just write

```python
for someKey in someDict:
```

rather than

```python
for someKey in someDict.keys():
```

because they have a similar meaning, but the latter creates an unnecessary object.

Similarly, when testing if a key is in a dictionary, just write

```python
if someKey in someDict:
```

rather than

```python
if someKey in someDict.keys():
```
### Summary of Dictionary Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Result</th>
<th>Mutates dict?</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.keys()</code></td>
<td>Returns all keys as a <code>dict_keys</code> object</td>
<td>No</td>
</tr>
<tr>
<td><code>.values()</code></td>
<td>Returns all values as a <code>dict_values</code> object</td>
<td>No</td>
</tr>
<tr>
<td><code>.items()</code></td>
<td>Returns (key, value) pairs as a <code>dict_items</code> object</td>
<td>No</td>
</tr>
<tr>
<td><code>.get(key [, val])</code></td>
<td>Returns corresponding value if <code>key</code> in dict, else returns <code>val</code>. The notation <code>[ , val]</code> means that the second argument <code>val</code> is optional and can be omitted. If it is not specified, it defaults to <code>None</code>.</td>
<td>No</td>
</tr>
<tr>
<td><code>.pop(key)</code></td>
<td>Removes key:val pair with given <code>key</code> from dict and returns associated val. Signals <code>Key Error</code> if key not in dict.</td>
<td>Yes</td>
</tr>
<tr>
<td><code>.update(dict2)</code></td>
<td>Adds new key:value pairs from <code>dict2</code> to <code>dict</code>, replacing any key:value pairs with existing key.</td>
<td>Yes</td>
</tr>
<tr>
<td><code>.clear()</code></td>
<td>Removes all items from the dict.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Image Source: (http://cs111.wellesley.edu/spring19)
Set Methods Summary

- `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 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.
Heads Up for Lab: Assert

• Python’s `assert` statement is a debugging aid that tests a condition.

• If the condition is true, it does nothing and your program just continues to execute.

• But if the `assert` condition evaluates to false, it raises an `AssertionError` exception with an optional error message.

• Assertions are internal self-checks for your program.

```
assertStatement = "assert" exp1 [""," exp2]
```

• `exp1` is the condition we test, and the optional `exp2` is an error message that's displayed if the assertion fails.
Heads Up for Lab: Isograms

- A word or phrase that has no repeated letter (no duplicates!)

```python
>>> isogram('ambidextrously')
True

>>> isogram('DOCTORWHO')
False

>>> isogram('uncopyrightable')
True
```
Acknowledgments

These slides have been adapted from:

- [http://cs111.wellesley.edu/spring19](http://cs111.wellesley.edu/spring19) and