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?

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

Python Collections

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

Type	Description	Example
list	a mutable sequence of arbitrary objects	`[-100, "blue", (1, 10), True]`
tuple	an immutable sequence of arbitrary objects	`(2017, "Mar", 2)`
string	an immutable sequence of characters	`"Go Wellesley!"`
range	an immutable sequence of numbers	`range(3)`
set	a mutable unordered collection of distinct objects.	`{1, 4, 5, 23}`
dict	a mutable unordered collection of key:value pairs, where keys are **immutable** and values are any Python objects	`{"orange": "fruit", 3: "March", "even": [2,4,6,8]}`

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

Properties of Sequences & Collections

- Collections (list, tuple, string, range, set, dict)
    - Find their length with `len()`
    - Check element membership in the collection with `in`
    - Are iterables (we can iterate over their elements in a loop)
- Sequences (list, tuple, string, range)
    - Use indices to access elements, e.g., `myList[2]`
    - Use slice operations for subsequences, e.g, `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

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

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]
```

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

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.

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

key

value

```
monthsLength = {
31: ['Jan','Mar','May','Jul','Aug','Oct','Dec'],
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 [3] cart['oreos'] = 3.99
In [4] cart['kiwis'] = 2.54
In [5] cart
Out [5] {'kiwis': 2.54, 'oreos': 3.99}
```

Note. keys may be listed in any order

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

```
In [6] dict([('Harry', 12), ('Hagrid', 40)]
Out [6] {'Harry': 12, 'Hagrid': 40}
```

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
```

Check Key Exists with `in` Operator

- 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
Out [6] False
```

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] daysOfMonth['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
```

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

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.

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

```
Out[164]: dict_keys(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
          'Oct', 'Nov', 'Dec'])
```

```
In [166]: daysOfMonth.values()
```

```
Out[166]: dict_values([31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31])
```

```
In [168]: daysOfMonth.items()
```

```
Out[168]: dict_items([('Jan', 31), ('Feb', 29), ('Mar', 31), ('Apr', 30), ('May', 3
          1), ('Jun', 30), ('Jul', 31), ('Aug', 31), ('Sep', 30), ('Oct', 31), ('No
          v', 30), ('Dec', 31)])
```

Iterating over/membership in Dicts

When iterating over the keys in a dictionary, just write

```
for someKey in someDict:
```

rather than

```
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

```
if someKey in someDict:
```

rather than

```
if someKey in someDict.keys():
```

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

Summary of Dictionary Methods

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

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!)

```
>>> isogram('ambidextrously')
True
>>> isogram('DOCTORWHO')
False
>>> isogram('uncopyrightable')
True
```

Acknowledgments

These slides have been adapted from:

- http://cs111.wellesley.edu/spring19 and
- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/