CS134:
Sets

Announcements & Logistics

- **HW 5** due today @ 10pm
- **Lab 4 Part 2** due Wednesday/Thursday 10pm
  - There is a Gradescope - Part 2 assignment
- **Midterm reminders:**
  - **Midterm Exam** is Thursday, October 17 at 6pm or 8pm in TPL203
    - **Midterm Review** is in place of class on Wednesday 10/16 during class, 9am-11:50am **Bring Questions!!**
    - **To Prepare**: *Redo:* [homework, **practice exams**, POGIL questions (including Application Questions), pre-labs & labs] w paper & pencil...then check your answers with Python!
- **Final Exam** schedule is posted: Wednesday, December 11 at 9:30am

Do You Have Any Questions?

Last Time: Scope

- **Scope**: variables, functions, objects have limited accessibility/visibility.
  - Understanding how this works helps us make decisions about where to define variables/functions/objects

Today's Plan

- **Sets** — a new data structure that allows us to efficiently store unordered collections of objects
- An example of designing algorithms to use sets

Sets

New Unordered Data Structure: Sets

- Lists are **ordered** collections of objects
- 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
- Elements 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"}
>>> empty_set = set() # empty set
```

New Unordered Data Structure: Sets

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

```
>>> first_choice = {'a', 'b', 'a', 'a', 'b', 'c'}
>>> uniques = set(first_choice)
>>> 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

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

Sets: Creating New Sets

- There are two ways to create a new set:
    - By placing curly brackets around elements:

```
>>> set_brack = {'aardvark'}
>>> set_brack
{'aardvark'}
```

    - By converting an iterable collection into a set:

```
>>> set_func = set('aardvark')
>>> set_func
{'d', 'v', 'a', 'r', 'k'}
```

Why letters here instead of the word?

Strings are iterable collection!

- And only one way to create an empty set:

```
>>> empty_set = set()
>>> empty_set
set()
```

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)
```

```
tulips
daisies
daffodils
asters
```

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 (foreshadowing...)

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

TypeError: unhashable type: 'list'
```

Set Operations

- The usual operations you think of in set theory are implemented as follows

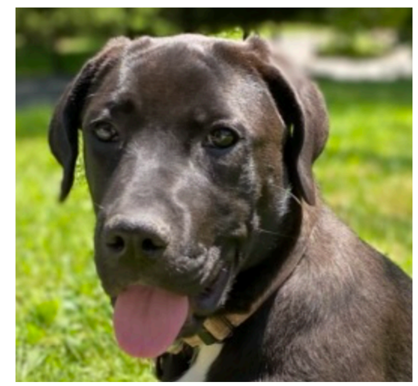
The following operations always return a new set.

- `s1 | s2` **(Set Union)**
  - Returns a new set that has all elements that are either in `s1` or `s2`
- `s1 & s2` **(Set Intersection)**
  - Returns a new set that has all the elements that are common to both sets.
- `s1 - s2` **(Set Difference)**
  - 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.

Set Operations

```
>>> cs134_dogs = {"wally", "pixel", "linus", "chelsea", "sally", "artie"}
```



```
>>> peanuts = {"sally", "linus", "charlie", "franklin", "lucy", "patty"}
```

Set Operations

```
>>> cs134_dogs = {"wally", "pixel", "linus", "chelsea", "sally", "artie"}
>>> peanuts = {"sally", "linus", "charlie", "franklin", "lucy", "patty"}
```

```
>>> union = cs134_dogs | peanuts
>>> union
{'sally', 'wally', 'patty', 'chelsea', 'pixel',
'franklin', 'lucy', 'artie', 'linus', 'charlie'}

>>> intersect = cs134_dogs & peanuts
>>> intersect
{'sally', 'linus'}

>>> diff = cs134_dogs - peanuts
>>> diff
{'chelsea', 'artie', 'wally', 'pixel'}

>>> cs134_dogs
{'sally', 'wally', 'linus', 'artie', 'chelsea', 'pixel'}
```

Original set is unchanged!

Set Operations: Mutators

```
>>> cs134_dogs = {"wally", "pixel", "linus", "chelsea", "sally", "artie"}
>>> peanuts = {"sally", "linus", "charlie", "franklin", "lucy", "patty"}
```

```
>>> cs134_dogs |= peanuts
>>> cs134_dogs
```
Original set is mutated!
```
{'sally', 'wally', 'patty', 'chelsea', 'pixel',
'franklin', 'lucy', 'artie', 'linus', 'charlie'}
```

```
>>> cs134_dogs = {"wally", "pixel", "linus", "chelsea", "sally", "artie"}
>>> cs134_dogs &= peanuts
>>> cs134_dogs
```
Original set is mutated!
```
{'sally', 'linus'}
```

```
>>> cs134_dogs = {"wally", "pixel", "linus", "chelsea", "sally", "artie"}
>>> cs134_dogs -= peanuts
>>> cs134_dogs
```
Original set is mutated!
```
{'wally', 'artie', 'chelsea', 'pixel'}
```

Set Operations

- The usual operations you think of in set theory are implemented as follows

The following operations always return a new set.

- `s1 | s2` **(Set Union)**
  - Returns a new set that has all elements that are either in `s1` or `s2`
- `s1 & s2` **(Set Intersection)**
  - Returns a new set that has all the elements that are common to both sets.
- `s1 - s2` **(Set Difference)**
  - 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.

Example: Word Puzzles

NYTimes Spelling Bee

The NYTimes Spelling Bee Puzzle is a source of interesting word problems. These words are spelled with an alphabet (called a "hive") of at most seven letters:

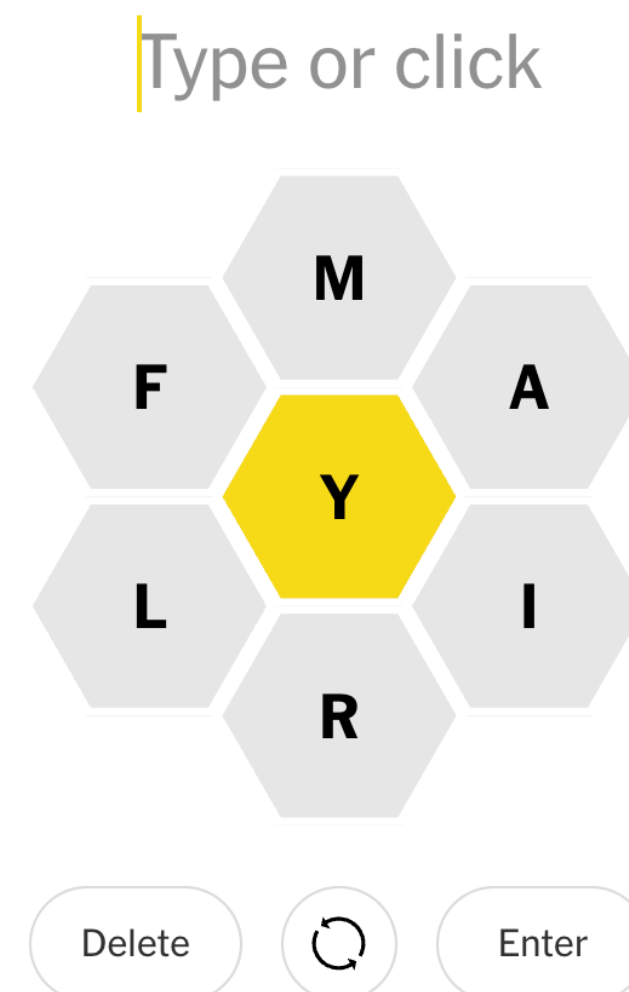
- https://www.nytimes.com/puzzles/spelling-bee

How many 4-7 letter isograms can we find in the 7 letters below, that all use the center letter?

The CS134 edition uses words that have letters that only appear once

(i.e., isograms)

```
'airy'
'army'
'fairly'
'firmly'
'ramify'
...
```



NYTimes Spelling Bee

- **Problem:** How many 4-7 letter isograms can we find in the 7 letters below, that all use `'y'`? `['y', 'm', 'a', 'i', 'f', 'l', 'r']`

- Possible algorithms to solve this:
  1. look at each word in the dictionary
  2. if it has in `'y'` it and a length greater than 3...
  3. ...and it's an isogram...
  4. ...and if it is only made up of the specified letters?
  5. Then it's a match

NYTimes Spelling Bee

- **Problem:** How many 4-7 letter isograms can we find in the 7 letters below, that all use `'y'`? `['y', 'm', 'a', 'i', 'f', 'l', 'r']`

```
# 1. look at each word in the dictionary
   # 2. if it has y in it and len > 3
   # 3. and it's an isogram
   # 4. and it's only made up of the specified letters
       # Add to our results list
```

NYTimes Spelling Bee

- **Problem:** How many 4-7 letter isograms can we find in the 7 letters below, that all use `'y'`? `['y', 'm', 'a', 'i', 'f', 'l', 'r']`

```
answers = []
for word in word_list:
    if 'y' in word and len(word) > 3:
        # 3. and it's an isogram
        # 4. and it's only made up of the specified letters
        answers += [word]
return answers
```

Helper Function!

NYTimes Spelling Bee

- **Problem:** How many 4-7 letter isograms can we find in the 7 letters below, that all use `'y'`? `['y', 'm', 'a', 'i', 'f', 'l', 'r']`

```
def is_isogram(word):
    """ Returns True if word is a string without any
    repeat letters
    >>> is_isogram("iris")
    False
    >>> is_isogram("lida")
    True
    """
   # How do we know if a word only has a unique number
   # of letters?
```

NYTimes Spelling Bee

- **Problem:** How many 4-7 letter isograms can we find in the 7 letters below, that all use `'y'`? `['y', 'm', 'a', 'i', 'f', 'l', 'r']`

```
def is_isogram(word):
    """ Returns True if word is a string without any
    repeat letters
    >>> is_isogram("iris")
    False
    >>> is_isogram("lida")
    True
    """
    return len(word) == len(set(word))
```

Take advantage of set(..) and how it only retains *unique* elements in a collection!

NYTimes Spelling Bee

- **Problem:** How many 4-7 letter isograms can we find in the 7 letters below, that all use `'y'`? `['y', 'm', 'a', 'i', 'f', 'l', 'r']`

```
answers = []
for word in word_list:
    if 'y' in word and len(word) > 3 and is_isogram(word):
        # 4. and it's only made up of the specified letters
        answers += [word]
return answers
```

More sets!

Set Difference

```
>>> # set 1 smaller than set 2
>>> set('airy') - set('ymaiflr')
set()
>>> # set 1 more letters than set 2
>>> set('maniacal') - set('ymaiflr')
{'n', 'c'}
>>> # set 1 same len as set 2
>>> set('bngepst') - set('ymaiflr')
{'b', 'n', 'g', 'e', 'p', 's', 't'}
>>> # set 1 same letters as set 2
>>> set('iflramy') - set('ymaiflr')
set()
```

If this difference operation results in an empty set, then the word is in the hive!

NYTimes Spelling Bee

- **Problem:** How many 4-7 letter isograms can we find in the 7 letters below, that all use `'y'`? `['y', 'm', 'a', 'i', 'f', 'l', 'r']`

```
answers = []
for word in word_list:
   if 'y' in word and len(word) > 3 and is_isogram(word)
    and not(set(word)-set(hive)):
      answers += [word]
return answers
```

Let's make it a function that uses arguments,
for generalizability!

NYTimes Spelling Bee

- **Problem:** How many 4-7 letter isograms can we find in the 7 letters below, that all use `'y'`? `['y', 'm', 'a', 'i', 'f', 'l', 'r']`

```
def spelling_bee(center, hive, word_list):
    answers = []
    for word in word_list:
        if center in word and len(word) > 3 and
          is_isogram(word) and not(set(word)-set(hive)):
            answers += [word]
    return answers
```

We need to call this function somewhere...

NYTimes Spelling Bee

- **Problem:** How many 4-7 letter isograms can we find in the 7 letters below, that all use `'y'`? `['y', 'm', 'a', 'i', 'f', 'l', 'r']`

```
def spelling_bee(center, hive, word_list):
    answers = []
    for word in word_list:
        if center in word and len(word) > 3 and
         is_isogram(word) and not(set(word)-set(hive)):
            answers += [word]
    return answers
```

```
if __name__ == '__main__': # only runs when code is run as a script

   # How many 4-7 letter isograms are in the letters below,
   using the letter 'y'?   ['y', 'm', 'a', 'i', 'f', 'l', 'r']
    spelling_bee('y', 'ymaiflr', read_words("/usr/share/dict/words"))
```

Helper function that reads in words from /usr/share/dict/words

The end!