CSE 160 Section 7

Set & Sorting

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Logistics

Due 11/14: HW4 Part 2

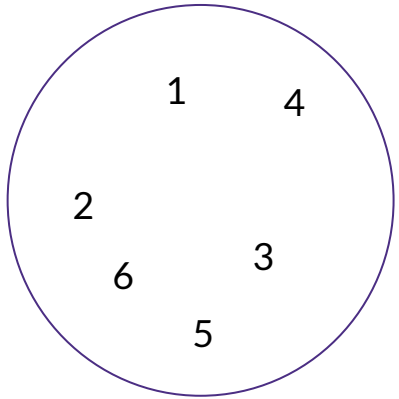
Due 11/16: HW5 Checkin (On Gradescope!)

Set & Sorting Review

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Sets

- Sets are a type of data structure which is unordered and unindexed
- There can be no duplicates
- Imagine it as a bag of values
- You can imagine a set that contains the values 1 through 6 like this:



What can be added into a set?

- Although you can convert any data structure into a set, you can only add immutable types into a set (just like dict keys)
- Data types that can not go in a set (mutable types)
  - Dictionaries
  - Lists
  - other sets
- Data types that can go in a set (immutable types)
  - Integers
  - Floats
  - Booleans (but why would you do this?)
  - Strings
  - Tuples

Property of sets

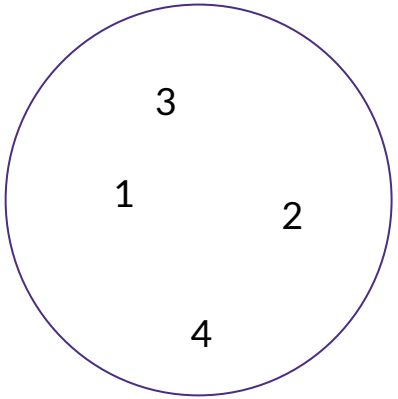
- All of the following code would make a set called s with the elements 1-4 inside

```
s = {1, 2, 3, 4}
```

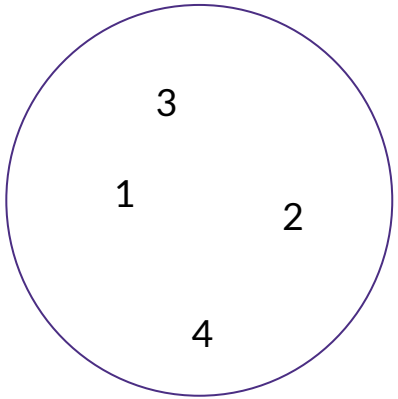
```
s = {1, 1, 1, 1, 1, 1, 2, 3, 4}
```

```
s = set([1, 2, 3, 4])
```

```
s = set()
s.add(1)
s.add(2)
s.add(3)
s.add(4)
```



Example



- To see all elements in a set, we can loop through it

```
for element in s:
    print(element)
```

- To see all elements in a set, we can loop through it

```
2 in s
```

Returns True

```
6 in s
```

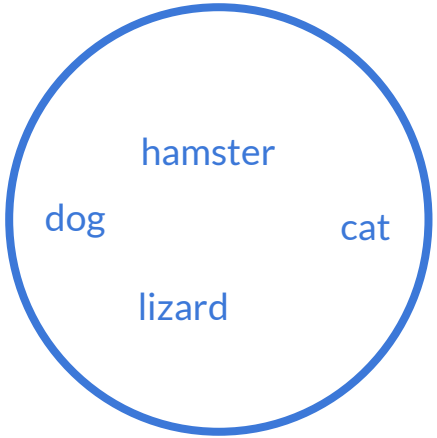
Returns False

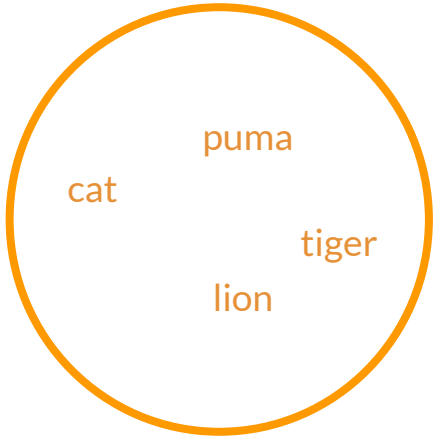
Looping through a Set

- **Say we have two sets:**

  pets = {“dog”, “hamster”, “lizard”, “cat”}

  felines = {“cat”, “puma”, “lion”, “tiger”}



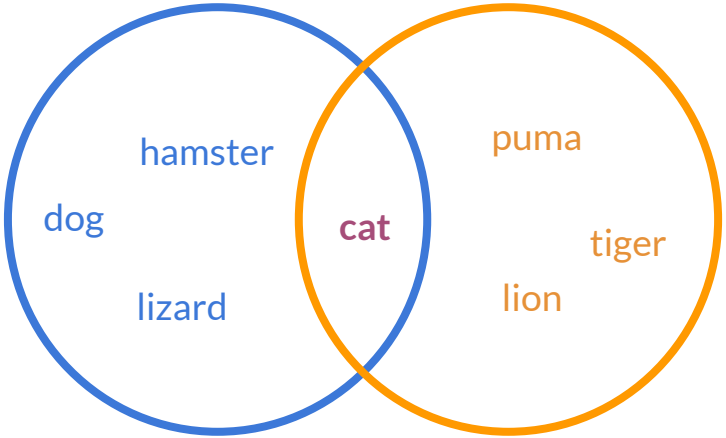


Example

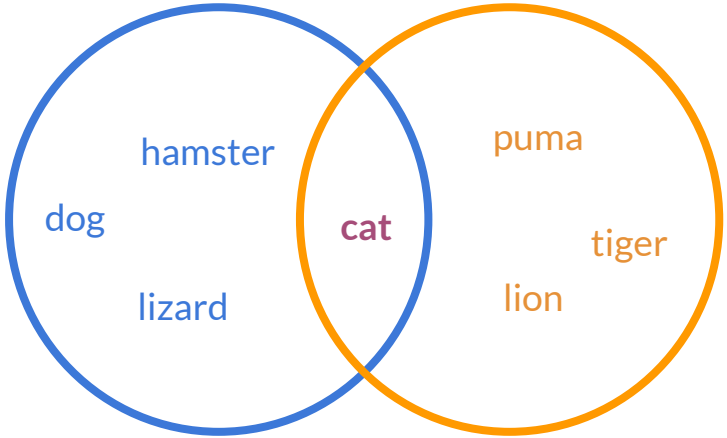
- **Say we have two sets:**

  pets = {“dog”, “hamster”, “lizard”, “cat”}

  felines = {“cat”, “puma”, “lion”, “tiger”}



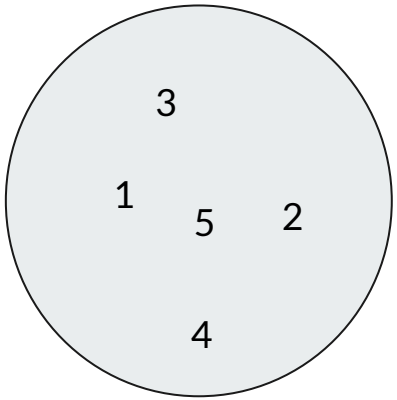
Example



- `pets | felines`
  - {“dog”, “hamster”, “lizard”, “cat”, “puma”, “lion”, “tiger”}
- `pets & felines`
  - {“cat”}
- `pets - felines`
  - {“dog”, “hamster”, “lizard”}
- `pets ^ felines`
  - {“dog”, “hamster”, “lizard”, “puma”, “lion”, “tiger”}

Add, Remove, Discard

Say we have the set s that has elements 1, 2, 3, 4 inside



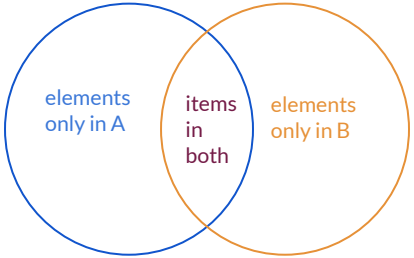
- Add
  - adds an element to the set
  - `s.add(5)`
- Remove
  - takes out an existing element from the set (Must exist in the set!)
  - `s.remove(5)`
- Discard
  - takes out an element from the set (doesn't need to be in there already)
  - `s.discard(5)`
- Pop
  - Returns a random element
  - `s.pop()`
  - Could return 1, 2, 3, or 4

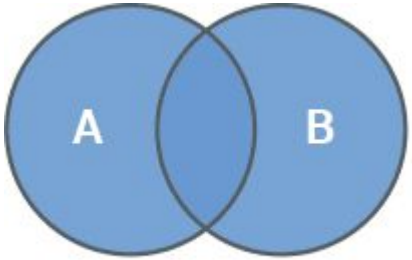
Add, Remove, Discard

Say we have the set s that has elements 1, 2, 3, 4 inside

- Add
  - adds an element to the set
  - s.add(5)
- Remove
  - takes out an existing element from the set (Must exist in the set!)
  - s.remove(5)
- Discard
  - takes out an element from the set (doesn't need to be in there already)
  - s.discard(5)
- Pop
  - Returns a random element
  - s.pop()
  - Could return 1, 2, 3, or 4

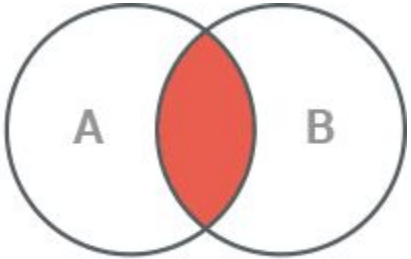
Set Operations





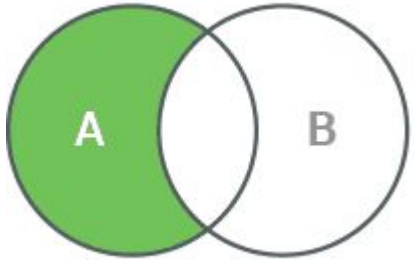
Union

A | B



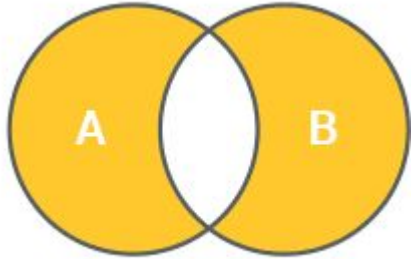
Intersection

A & B



Difference

A - B



Symmetric Difference

A ^ B

Tuple Review

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Tuples

- **Tuple is a collection which is ordered and unchangeable.**
- **Lists and tuples are similar, but have different properties**
- **The table at the right shows what kind of things you can do with a tuple, but not a list.**
- **Let the data structure be called name. A ✅ means you can do it, a 🚫 means it won't work**

Description	Example	list	tuple
indexing	`name[i]`	✅	✅
negative indexing	`name[-i]`	✅	✅
slicing	`name[i:j]`	✅	✅
checking if item exists	`item in name`	✅	✅
looping	`for item in name`	✅	✅
length	`len(name)`	✅	✅
changing items	`name[i] = item`	✅	🚫
appending items	`name.append(item)`	✅	🚫
put in a set	`set().add(name)`	🚫	✅
use a dict keys	`dict(name:val)`	🚫	✅

Making a Tuple

These are all ways to make tuples:

```
t = (1, 2, 3)
```

```
t = tuple([1, 2, 3])
```

```
t = (1,)
```

Note that to make a one element tuple you need to add a comma after the one value! `(1)` would not work!

Itemgetter Review

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Why do we care about Itemgetter

- Very easy way to sort values on multiple attributes
- Way to sort dictionaries into a list
  - if my_dict is a dictionary, then list(my_dict.items()) will return a list of key value tuple pairs!
- Super important for hw5!

Itemgetter

- Itemgetter returns a function

```
from operator import itemgetter

get_3rd_item = itemgetter(2)
get_3rd_item([7, 3, 8])          -> 8
# this is the same as
itemgetter(2)([7, 3, 8])         -> 8
```

Sorting with Itemgetter

- Useful for when you have a list of tuples where every item inside those tuples corresponds to a particular feature
  - ex: the element at index 0 of every tuple is a name, the element at index 1 of every tuple is an age)
  - lst = [('Anne', 5), ('Bob', 6), ('Carl', 3), ('Elisa', 2), ('Diana', 2)]

- If you want to sort with the feature in the tuple at the index i, do so like this:
  - sorted_lst = sorted(lst, key = itemgetter(i))

- Sort by the “least important” feature first (ie a tie breaker), the more important features last

Sorting with Itemgetter

Using itemgetter, define a function called find_oldest that takes in a list of tuples in the form (name, age) and returns a list of tuples belonging to the oldest people.

For example, given age_list = [("Tom", 19), ("Max", 26), ("James", 12), ("Carol", 10)], find_oldest(age_list) would return [("Max", 26)]

Sorting with Itemgetter

```
lst = [ ("Tom", 19), ("Max", 26), ("James", 12), ("Carol", 10) ]

# sort by age
sort_age = sorted(lst, key = itemgetter(1), reverse=True)

oldest_age = sort_age[0][1]
ret_list = []

for pair in sort_age:
    if(pair[1] == oldest_age):
        ret_list.append(pair)
    else:
        return ret_list # return early since it is sorted
return ret_list
```

Sorting with Itemgetter

Using itemgetter, define a function called find_oldest that takes in a list of tuples in the form (name, age) and returns a list of tuples belonging to the oldest people. If there is a tie, return a list of the names and ages of the people sharing the same (oldest) age in a new list in alphabetical order.

For example, given age_list = [("Tom", 19), ("Max", 26), ("James", 12), ("Alice", 26), ("Carol", 10)], find_oldest(age_list) would return [("Alice", 26), ("Max", 26)]

Sorting with Itemgetter

```
lst = [ ("Tom", 19), ("Max", 26), ("James", 12), ("Carol", 10) ]

# sort by age
sort_name = sorted(lst, key=itemgetter(0))
sort_age = sorted(sort_name, key = itemgetter(1), reverse=True)

oldest_age = sort_age[0][1]
ret_list = []

for pair in sort_age:
    if(pair[1] == oldest_age):
        ret_list.append(pair)
    else:
        return ret_list # return early since it is sorted
return ret_list
```

Kahoot

We'll use this kahoot to go over problems 1, 2, and 4

No pressure to win (as always)

Section Problems

We'll emphasize problems 5 and 6 (since they are the most relevant for the homework!)

Starter Code
(Problem 5)