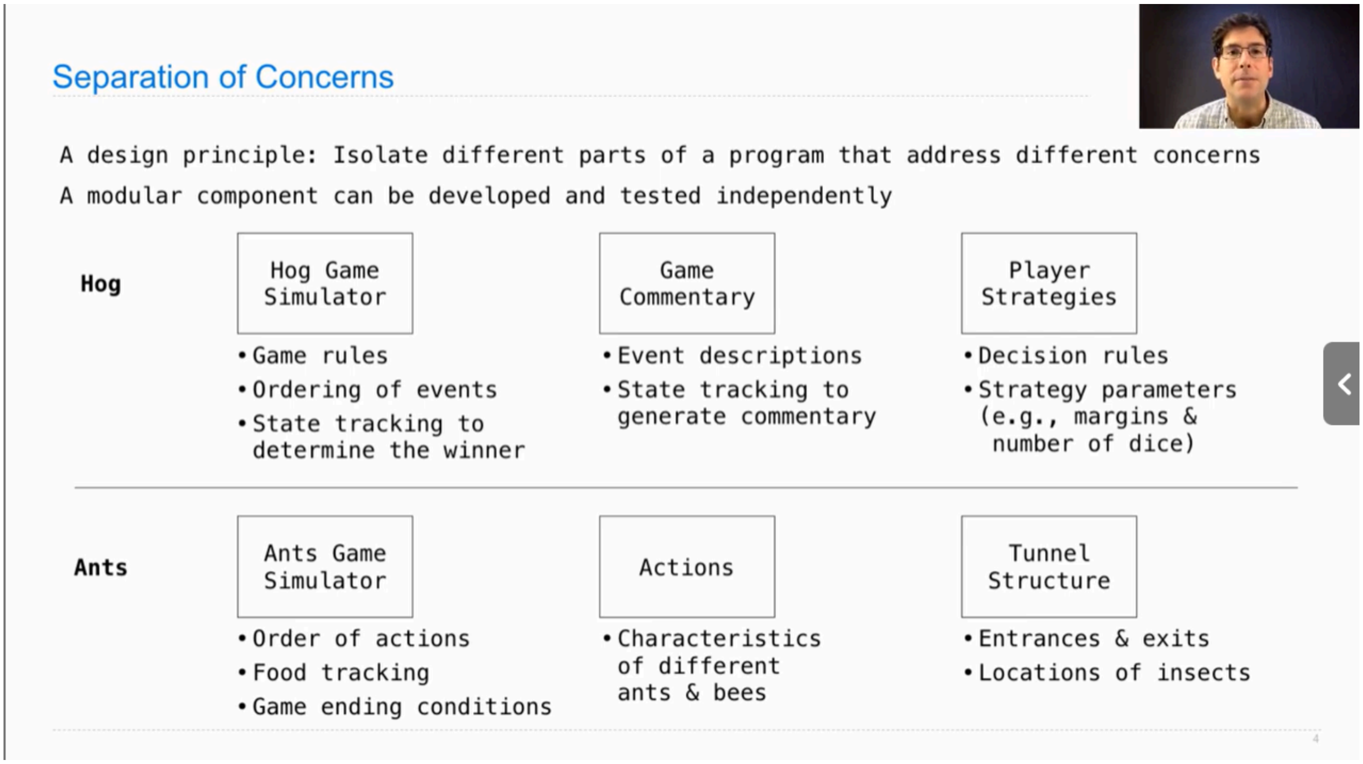
Lecture 23.Decomposition

Modular Design



Separation of Concerns

A design principle: Isolate different parts of a program that address different concerns

A modular component can be developed and tested independently

Hog

Hog Game Simulator
- Game rules
- Ordering of events
- State tracking to determine the winner

Game Commentary
- Event descriptions
- State tracking to generate commentary

Player Strategies
- Decision rules
- Strategy parameters (e.g., margins & number of dice)

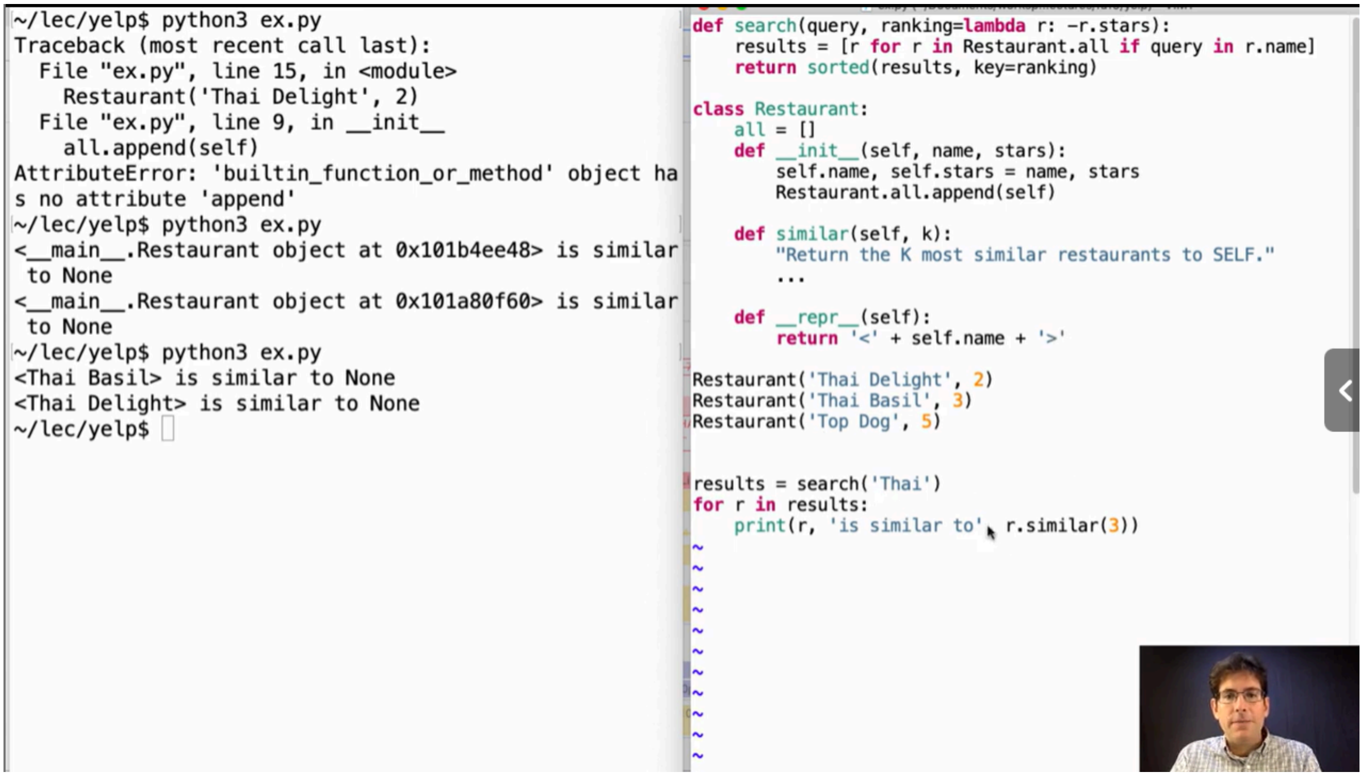
Ants

Ants Game Simulator
- Order of actions
- Food tracking
- Game ending conditions

Actions
- Characteristics of different ants & bees

Tunnel Structure
- Entrances & exits
- Locations of insects

Example: Restaurant Search Data



```
~/lec/yelp$ python3 ex.py
Traceback (most recent call last):
  File "ex.py", line 15, in <module>
    Restaurant('Thai Delight', 2)
  File "ex.py", line 9, in __init__
    all.append(self)
AttributeError: 'builtin_function_or_method' object ha
s no attribute 'append'
~/lec/yelp$ python3 ex.py
<__main__.Restaurant object at 0x101b4ee48> is similar
 to None
<__main__.Restaurant object at 0x101a80f60> is similar
 to None
~/lec/yelp$ python3 ex.py
<Thai Basil> is similar to None
<Thai Delight> is similar to None
~/lec/yelp$ 
```

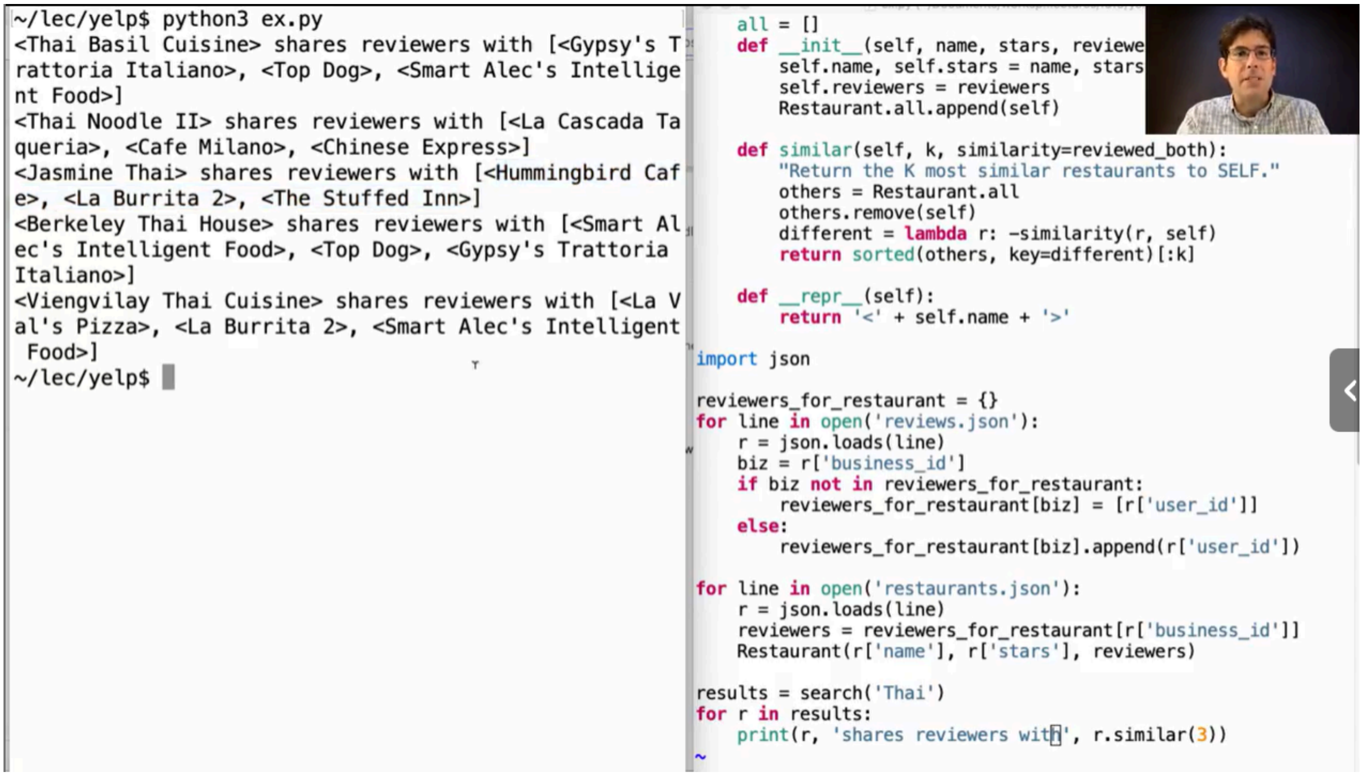
```python
def search(query, ranking=lambda r: -r.stars):
    results = [r for r in Restaurant.all if query in r.name]
    return sorted(results, key=ranking)

class Restaurant:
    all = []
    def __init__(self, name, stars):
        self.name, self.stars = name, stars
        Restaurant.all.append(self)

    def similar(self, k):
        "Return the K most similar restaurants to SELF."
        ...

    def __repr__(self):
        return '<' + self.name + '>'

Restaurant('Thai Delight', 2)
Restaurant('Thai Basil', 3)
Restaurant('Top Dog', 5)


results = search('Thai')
for r in results:
    print(r, 'is similar to', r.similar(3))
~
~
~
~
~
~
~
~
~
~
~
```

Implementation of the "simular” method

```
def similar(self, k, similarity):
    """Return the K most similar restaurants to SELF, using SIMILARITY for 
comparison"""
    others = list(Restaurant.all)
    others.remove(self)
    return sorted(others, key=lambda r:simuilarity(self, r), reverse=True)[:k]
```

introducing users

```
def reviewed_both(r, s):
    return len([x for x in r.reviewers if x in s.reviewers])
```

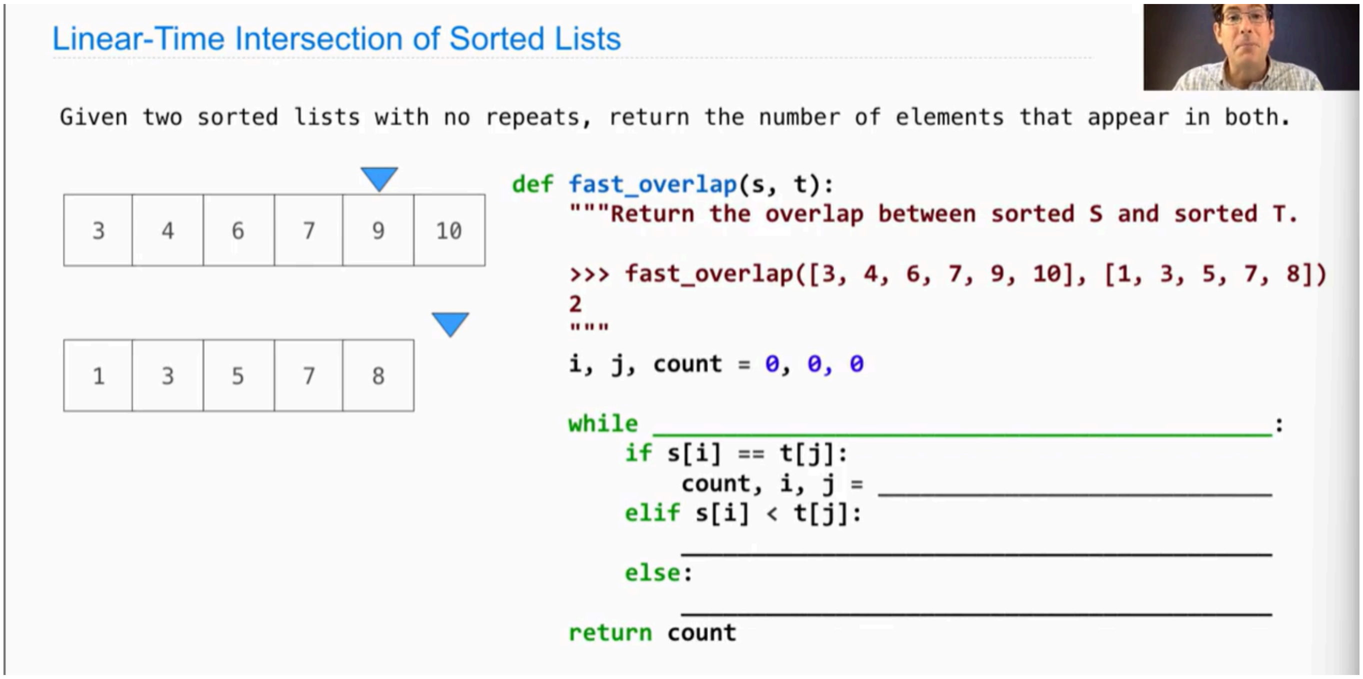


```
~/lec/yelp$ python3 ex.py
<Thai Basil Cuisine> shares reviewers with [<Gypsy's T
rattoria Italiano>, <Top Dog>, <Smart Alec's Intellige
nt Food>]
<Thai Noodle II> shares reviewers with [<La Cascada Ta
queria>, <Cafe Milano>, <Chinese Express>]
<Jasmine Thai> shares reviewers with [<Hummingbird Caf
e>, <La Burrita 2>, <The Stuffed Inn>]
<Berkeley Thai House> shares reviewers with [<Smart Al
ec's Intelligent Food>, <Top Dog>, <Gypsy's Trattoria
Italiano>]
<Viengvilay Thai Cuisine> shares reviewers with [<La V
al's Pizza>, <La Burrita 2>, <Smart Alec's Intelligent
 Food>]
~/lec/yelp$
```

```
    all = []
    def __init__(self, name, stars, reviewe
        self.name, self.stars = name, stars
        self.reviewers = reviewers
        Restaurant.all.append(self)

    def similar(self, k, similarity=reviewed_both):
        "Return the K most similar restaurants to SELF."
        others = Restaurant.all
        others.remove(self)
        different = lambda r: -similarity(r, self)
        return sorted(others, key=different)[:k]

    def __repr__(self):
        return '<' + self.name + '>'

import json

reviewers_for_restaurant = {}
for line in open('reviews.json'):
    r = json.loads(line)
    biz = r['business_id']
    if biz not in reviewers_for_restaurant:
        reviewers_for_restaurant[biz] = [r['user_id']]
    else:
        reviewers_for_restaurant[biz].append(r['user_id'])

for line in open('restaurants.json'):
    r = json.loads(line)
    reviewers = reviewers_for_restaurant[r['business_id']]
    Restaurant(r['name'], r['stars'], reviewers)

results = search('Thai')
for r in results:
    print(r, 'shares reviewers with', r.similar(3))
~
```

Improving the efficiency

reviewed_both takes too much time to seach



Linear-Time Intersection of Sorted Lists

Given two sorted lists with no repeats, return the number of elements that appear in both.

3	4	6	7	9	10

1	3	5	7	8

```
def fast_overlap(s, t):
    """Return the overlap between sorted S and sorted T.

    >>> fast_overlap([3, 4, 6, 7, 9, 10], [1, 3, 5, 7, 8])
    2
    """
    i, j, count = 0, 0, 0

    while ______________________________________:
        if s[i] == t[j]:
            count, i, j = ______________________________
        elif s[i] < t[j]:
            ____________________________________________
        else:
            ____________________________________________
    return count
```

my implementation

```
def fast_overlap(s, t):
    """Return the overlab between sorted S and sorted T"""
    i, j, count = 0, 0, 0
    while i < len(s) and j < len(t):
        if s[i] == t[j]:
            count, i, j = count+1, i+1, j+1
        elif s[i] < t[j]:
            i += 1
        else:
            j += 1
    return count
```

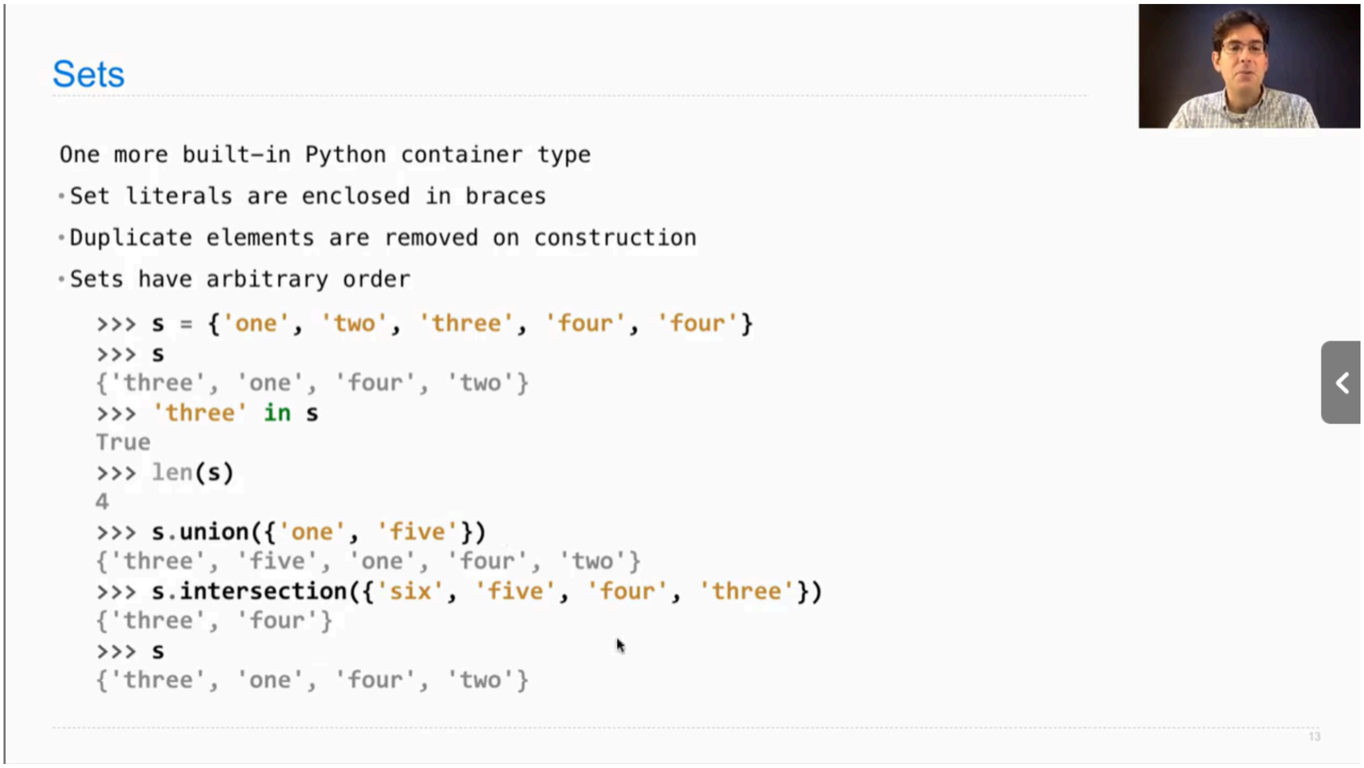
improved reviewed_both:

```
def reviewed_both:
    return fast_overlap(t.reviewers, s.reviewvers)
```

main loop

```
while True:
    print('>', end=' ')
    results = search(input().strip())
    for r in results:
        print(r, 'shares reviewers with', r.eimilar(3))
```

Sets



Sets

One more built-in Python container type

- Set literals are enclosed in braces
- Duplicate elements are removed on construction
- Sets have arbitrary order

```
>>> s = {'one', 'two', 'three', 'four', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
>>> 'three' in s
True
>>> len(s)
4
>>> s.union({'one', 'five'})
{'three', 'five', 'one', 'four', 'two'}
>>> s.intersection({'six', 'five', 'four', 'three'})
{'three', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
```