Decomposition
Announcements
Modular Design
Separation of Concerns
Separation of Concerns

A design principle: Isolate different parts of a program that address different concerns
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
**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 Commentary
- Player Strategies
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
- Game Commentary
- Player Strategies
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 Commentary
- Player Strategies

- Game rules
- Ordering of events
**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**

- **Player Strategies**
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

- Player Strategies
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

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

Game Commentary
- Event descriptions
- State tracking to generate commentary

Player Strategies
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
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)
**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.

<table>
<thead>
<tr>
<th>Hog Game Simulator</th>
<th>Game Commentary</th>
<th>Player Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Game rules</td>
<td>• Event descriptions</td>
<td>• Decision rules</td>
</tr>
<tr>
<td>• Ordering of events</td>
<td>• State tracking to generate commentary</td>
<td>• Strategy parameters (e.g., margins &amp; number of dice)</td>
</tr>
</tbody>
</table>

**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)
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
- Actions
- Tunnel Structure
## 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

- **Actions**

- **Tunnel Structure**
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

- Actions

- Tunnel Structure
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

- Tunnel Structure
## 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
### 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
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

<table>
<thead>
<tr>
<th>Hog Game Simulator</th>
<th>Game Commentary</th>
<th>Player Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Game rules</td>
<td>• Event descriptions</td>
<td>• Decision rules</td>
</tr>
<tr>
<td>• Ordering of events</td>
<td>• State tracking to generate commentary</td>
<td>• Strategy parameters (e.g., margins &amp; number of dice)</td>
</tr>
<tr>
<td>• State tracking to determine the winner</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ants Game Simulator</th>
<th>Actions</th>
<th>Tunnel Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Order of actions</td>
<td>• Characteristics of different ants &amp; bees</td>
<td>• Entrances &amp; exits</td>
</tr>
<tr>
<td>• Food tracking</td>
<td></td>
<td>• Locations of insects</td>
</tr>
<tr>
<td>• Game ending conditions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: Restaurant Search
Restaurant Search Data

Given the following data, look up a restaurant by name and show related restaurants.
Restaurant Search Data

Given the following data, look up a restaurant by name and show related restaurants.

{"business_id": "gclB3ED6uk6viW1olSb_uA", "name": "Cafe 3", "stars": 2.0, "price": 1, ...}
Restaurant Search Data

Given the following data, look up a restaurant by name and show related restaurants.

{"business_id": "gclB3ED6uk6viWlolSb_uA", "name": "Cafe 3", "stars": 2.0, "price": 1, ...}
{"business_id": "WXKx2I2SEzBpeUGtDMCS8A", "name": "La Cascada Taqueria", "stars": 3.0, "price": 2}
Given the following data, look up a restaurant by name and show related restaurants.

{"business_id": "gclB3ED6uk6viWlolSb_uA", "name": "Cafe 3", "stars": 2.0, "price": 1, ...}
{"business_id": "WXKx2I2SEzBpeUGtDMCS8A", "name": "La Cascada Taqueria", "stars": 3.0, "price": 2}
Restaurant Search Data

Given the following data, look up a restaurant by name and show related restaurants.

{"business_id": "gclB3ED6uk6viW1olSb_uA", "name": "Cafe 3", "stars": 2.0, "price": 1, ...}
{"business_id": "WXKx2I2SEzBpeUGtDMCS8A", "name": "La Cascada Taqueria", "stars": 3.0, "price": 2}
...
{"business_id": "gclB3ED6uk6viW1olSb_uA", "user_id": "xVocUszkZtAqCgxWak3xVQ", "stars": 1, "text": "Cafe 3 (or Cafe Tre, as I like to say) used to be the bomb diggity when I first lived in the dorms but sadly, quality has dramatically decreased over the years....", "date": "2012-01-19", ...}
Given the following data, look up a restaurant by name and show related restaurants.

{"business_id": "gclB3ED6uk6viWlo1Sb_uA", "name": "Cafe 3", "stars": 2.0, "price": 1, ...}
{"business_id": "WXKx2I2SEzBpeUGtDMCS8A", "name": "La Cascada Taqueria", "stars": 3.0, "price": 2}
...
{"business_id": "gclB3ED6uk6viWlo1Sb_uA", "user_id": "xVocUszkZtAqCxgWak3xVQ", "stars": 1, "text": "Cafe 3 (or Cafe Tre, as I like to say) used to be the bomb diggity when I first lived in the dorms but sadly, quality has dramatically decreased over the years....", "date": "2012-01-19", ...}
{"business_id": "WXKx2I2SEzBpeUGtDMCS8A", "user_id": "84dCHkhWGl0tk30VvaY5A", "stars": 2, "text": "-Excuse me for being a snob but if I wanted a room temperature burrito I would take one home, stick it in the fridge for a day, throw it in the microwave for 45 seconds, then eat it. NOT go to a resturant and pay like seven dollars for one...", "date": "2009-04-30", ...}
Restaurant Search Data

Given the following data, look up a restaurant by name and show related restaurants.

{"business_id": "gclB3ED6uk6viW1olSb_uA", "name": "Cafe 3", "stars": 2.0, "price": 1, ...
{"business_id": "WXKx2I2SEzBpeUGtDMCS8A", "name": "La Cascada Taqueria", "stars": 3.0, "price": 2} ...
{"business_id": "gclB3ED6uk6viW1olSb_uA", "user_id": "xVocUszkZtAqCgwak3xVQ", "stars": 1, "text": "Cafe 3 (or Cafe Tre, as I like to say) used to be the bomb diggity when I first lived in the dorms but sadly, quality has dramatically decreased over the years....", "date": "2012-01-19", ...
{"business_id": "WXKx2I2SEzBpeUGtDMCS8A", "user_id": "84dcKhW8IDtk3OVvaY5A", "stars": 2, "text": "Excuse me for being a snob but if I wanted a room temperature burrito I would take one home, stick it in the fridge for a day, throw it in the microwave for 45 seconds, then eat it. NOT go to a resturant and pay like seven dollars for one...", "date": "2009-04-30", ...
...
Restaurant Search Data

Given the following data, look up a restaurant by name and show related restaurants.

{"business_id": "gclB3ED6uk6viW1olSb_uA", "name": "Cafe 3", "stars": 2.0, "price": 1, ...}
{"business_id": "WXKx2I2SEzBpeUGtDMCS8A", "name": "La Cascada Taqueria", "stars": 3.0, "price": 2}
...
{"business_id": "gclB3ED6uk6viW1olSb_uA", "user_id": "xVocUszkZtAqCxgWak3xVQ", "stars": 1, "text": "Cafe 3 (or Cafe Tre, as I like to say) used to be the bomb diggity when I first lived in the dorms but sadly, quality has dramatically decreased over the years....", "date": "2012-01-19", ...}
{"business_id": "WXKx2I2SEzBpeUGtDMCS8A", "user_id": "84dCHkhWG8IDtk30VvaY5A", "stars": 2, "text": "-Excuse me for being a snob but if I wanted a room temperature burrito I would take one home, stick it in the fridge for a day, throw it in the microwave for 45 seconds, then eat it. NOT go to a resturant and pay like seven dollars for one...", "date": "2009-04-30", ...}
...

(Demo)
Example: Similar Restaurants
Discussion Question: Most Similar Restaurants
Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher `similarity` values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the `similarity` function, but not containing `self`. 
Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher `similarity` values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the `similarity` function, but not containing `self`.

```python
def similar(self, k, similarity):
```
Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher `similarity` values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the `similarity` function, but not containing `self`.

```python
def similar(self, k, similarity):
    "Return the K most similar restaurants to SELF, using SIMILARITY for comparison."
```
Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher `similarity` values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the `similarity` function, but not containing `self`.

```python
def similar(self, k, similarity):
    "Return the K most similar restaurants to SELF, using SIMILARITY for comparison."
    others = list(Restaurant.all)
```

Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher `similarity` values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the `similarity` function, but not containing `self`.

```python
def similar(self, k, similarity):
    "Return the K most similar restaurants to SELF, using SIMILARITY for comparison."

    others = list(Restaurant.all)

    others.______________(___________)
```
Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher `similarity` values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the `similarity` function, but not containing `self`.

```python
def similar(self, k, similarity):
    "Return the K most similar restaurants to SELF, using SIMILARITY for comparison."

    others = list(Restaurant.all)

    others.______________(______________)

    return sorted(others, key=______________________________)_____________________
```
Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher `similarity` values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the `similarity` function, but not containing `self`.

```python
def similar(self, k, similarity):
    "Return the K most similar restaurants to SELF, using SIMILARITY for comparison."

    others = list(Restaurant.all)

    others.______________(______________)

    return sorted(others, key=____________________________)(________________________)
```

**sorted**(iterable, /, *, key=None, reverse=False)
Return a new list containing all items from the iterable in ascending order. A custom key function can be supplied to customize the sort order, and the reverse flag can be set to request the result in descending order.
Discussion Question: Most Similar Restaurants

Implement `similar`, a Restaurant method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher `similarity` values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the `similarity` function, but not containing `self`.

```python
def similar(self, k, similarity):
    "Return the K most similar restaurants to SELF, using SIMILARITY for comparison."

    others = list(Restaurant.all)

    others.___remove___(______________)

    return sorted(others, key=_______________________________)________________
```

`sorted(iterable, /, *, key=None, reverse=False)`
Return a new list containing all items from the iterable in ascending order. A custom key function can be supplied to customize the sort order, and the reverse flag can be set to request the result in descending order.
Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher `similarity` values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the `similarity` function, but not containing `self`.

```python
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=_______________________________)____________________
```

`sorted(iterable, /, *, key=``None``, reverse=``False``)
Return a new list containing all items from the iterable in ascending order. A custom key function can be supplied to customize the sort order, and the reverse flag can be set to request the result in descending order.`
Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher `similarity` values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the `similarity` function, but not containing `self`.

```python
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: -similarity(self, r)____)__________________
```

`sorted(iterable, /, *, key=None, reverse=False)`
Return a new list containing all items from the iterable in ascending order. A custom key function can be supplied to customize the sort order, and the reverse flag can be set to request the result in descending order.
Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher `similarity` values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the `similarity` function, but not containing `self`.

```python
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: -similarity(self, r)____)[:k]
```

---

`sorted(iterable, /, *, key=None, reverse=False)`
Return a new list containing all items from the iterable in ascending order. A custom key function can be supplied to customize the sort order, and the reverse flag can be set to request the result in descending order.
Example: Reading Files

(Demo)
Set Intersection
Linear-Time Intersection of Sorted Lists

Given two sorted lists with no repeats, return the number of elements that appear in both.
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 |
Linear-Time Intersection of Sorted Lists

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

- List 1: 3 4 6 7 9 10
- List 2: 1 3 5 7 8

Number of common elements: 2
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
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
```
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 |
```
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
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
```
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
```
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
```
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 |
```
Linear-Time Intersection of Sorted Lists

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

```
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
```
Linear-Time Intersection of Sorted Lists

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

```
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 i < len(s) and j < len(t):
        if s[i] == t[j]:
            count, i, j = ________________________________
        elif s[i] < t[j]:
            ________________________________
        else:
            ________________________________
    return count
```
Linear-Time Intersection of Sorted Lists

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

```python
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 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]:
            ____________________________
        else:
            ____________________________
    return count
```
Linear-Time Intersection of Sorted Lists

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

```python
def fast_overlap(s, t):
    """Return the overlap 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 = i + 1
        else:
            j = j + 1
    return count

>>> fast_overlap([3, 4, 6, 7, 9, 10], [1, 3, 5, 7, 8])
2
"""
```

```python
def fast_overlap(s, t):
    """Return the overlap 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 = i + 1
        else:
            j = j + 1
    return count

>>> fast_overlap([3, 4, 6, 7, 9, 10], [1, 3, 5, 7, 8])
2
"""
```
Linear-Time Intersection of Sorted Lists

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

```python
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 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 = i + 1
        else:
            j = j + 1
    return count
```
Linear-Time Intersection of Sorted Lists

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

```python
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 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 = i + 1
        else:
            j = j + 1
    return count
```

(Demo)
Sets
Sets
Sets

One more built-in Python container type
Sets

One more built-in Python container type

- Set literals are enclosed in braces
Sets

One more built-in Python container type

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

One more built-in Python container type

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

One more built-in Python container type

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

```python
>>> s = {'one', 'two', 'three', 'four', 'four'}
```
Sets

One more built-in Python container type

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

```python
>>> s = {'one', 'two', 'three', 'four', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
```
Sets

One more built-in Python container type

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

```python
>>> s = {'one', 'two', 'three', 'four', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
>>> 'three' in s
True
```
Sets

One more built-in Python container type

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

```python
>>> s = {'one', 'two', 'three', 'four', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
>>> 'three' in s
True
>>> len(s)
4
```
Sets

One more built-in Python container type

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

```python
>>> s = {'one', 'two', 'three', 'four', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
>>> 'three' in s
True
>>> len(s)
4
```
Sets

One more built-in Python container type

• Set literals are enclosed in braces

• Duplicate elements are removed on construction

• Sets have arbitrary order

```python
>>> 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'}
```
Sets

One more built-in Python container type

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

```python
>>> 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'}
```