CS134 Lecture 15:

Sets

Announcements & Logistics

- No HW due next Monday
- **Midterm reminders:**
  - **Review:** **Monday 3/11** from 7-9pm
  - **Exam** **Thurs 3/14** from 6-7:30pm OR 8-9:30pm
    - Both exam and review are in Bronfman Auditorium
    - Exam only includes material up to this week
    - Sample Exam posted!
- New Instructor Help Hours Schedule
  - Wednesday 1-4, Thursday 1-4

Do You Have Any Questions?

Last Time

- Describe how scope works when lists are passed as function parameters (interaction between scope and aliasing)
- Explore two new Python types:
    - tuples: *immutable ordered* alternative to lists

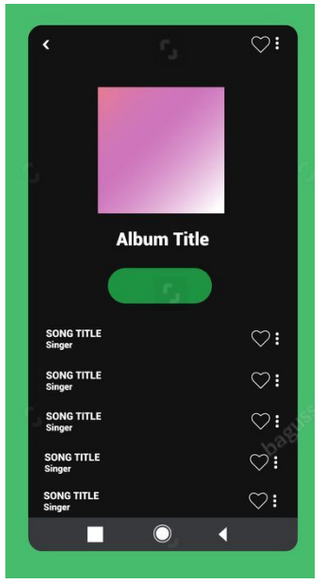
Today's Plan

- Explore another new Python type:
    - sets: *mutable unordered* collection
- Use tuples and sets in example functions

Sets

New Unordered Data Structure: Sets

- Lists and tuples both are ***ordered collections***
    - Order here refers to **numerical indices** to identify item position
- Sometimes there is no inherent numerical ordering of a collection, e.g.
    - Items in a grocery cart
    - Collection of songs on Spotify
- For **unordered collections**, we care the most about:
    - No duplicates
    - Membership: what is in the collection, what is not



Sets: Syntax and Properties

- Sets are written as **comma separated values** between curly braces **{}**
- Elements in a set must be **unique** and **immutable**
    - No mutable type allowed as an item of a set
    - No duplicates in a set

```
nums = {42, 17, 8, 57, 23}
flowers = {"tulips", "daffodils", "asters", "daisies"}
empty_set = set() # empty set
```

Sets: Syntax and Properties

- Sets are written as **comma separated values** between curly braces **{}**
- Elements in a set must be **unique** and **immutable**
    - No mutable type allowed as an item of a set
    - **No duplicates** in a set

```
# what if make a set with duplicates?
dup_set = {1, 1, 2, 2, 2, 3, 4, 5, 5, 5}


# what is in dup_set?
dup_set
```

```
{1, 2, 3, 4, 5}
```

No duplicates!

Sets: Syntax and Properties

- Sets are written as **comma separated values** between curly braces **{}**
- Elements in a set must be **unique** and **immutable**
    - **No mutable type allowed** as an item of a set
    - No duplicates in a set

```
# will this work?

l_set = {[1, 2, 3], "hello"}
```

Only immutable items in a set

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[12], line 3
      1 # will this work?
----> 3 l_set = {[1, 2, 3], "hello"}

TypeError: unhashable type: 'list'
```

Sets: Properties Overview

- Sets are ***mutable***, **unordered** collections of **immutable** objects
  - Sets can change (e.g., we can add and remove items)
  - Sets have no order
  - Sets cannot contain mutable types
- **Important**: Sets can be useful way of **eliminating duplicate values**

```
print(set("aabrakadabra"))

{'a', 'k', 'r', 'b', 'd'}
```

Loses ordering!

Potential **downside** of removing duplicates from a sequence using a set?

Tuples as Immutable Sequences

- Tuples, like strings, support any sequence operation that ***does not involve mutation***: e.g,
  - `len()` function: returns number of elements in tuple
  - `[]` indexing: access specific element
  - `+, *`: tuple concatenation
  - `[:]`: slicing to return subset of a tuple (as a new tuple)
  - `in` and `not in`: check membership of an object in a tuple
  - `for-loops`: iterate over elements in tuple (in order)

Sets: Properties Overview

- Sets support some familiar operators, functions and iteration patterns:
  - `len()`: returns number of items in a set
  - `in` and `not in`: check membership of an item in a set
  - `for-loops`: iterate over items in set (in arbitrary order)

Sets are Unordered

- 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'
```

Sets: Creating New Sets

- There are two ways to create a new set:
    - By placing curly brackets around elements
    - By using the built-in `set()` function
- And only one way to create an empty set

```
emp_set = set()
```

Can't write `emp_set = {}` which creates a different data type (empty dictionary)

Set Operations

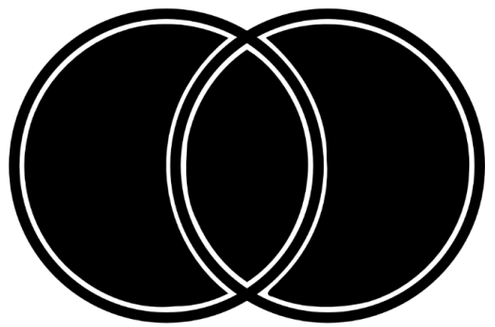
- The usual operations you think of in set theory are implemented as follows

The following always return a new set.

- `s1 | s2` **(Set Union)**

  - Returns a new set that has all elements that are either in `s1` or `s2`

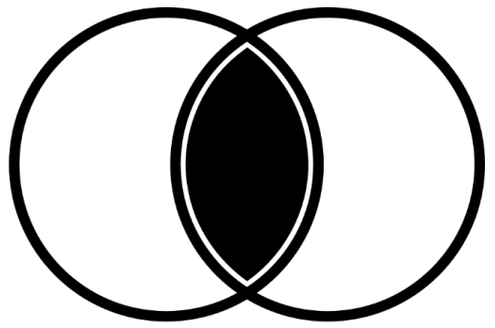
Union



- `s1 & s2` **(Set Intersection)**

  - Returns a new set that has all the elements that are common to both sets.

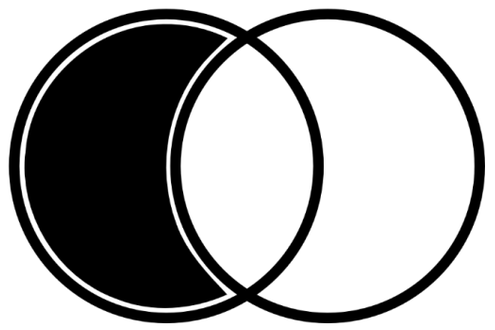
Intersection



- `s1 - s2` **(Set Difference)**

  - Returns a new set that has all the elements of `s1` that are not in `s2`

Difference



Sets are Mutable

- Sets are a **mutable data type**
  - There exists "methods" to mutate sets, such as `.add()`, `.remove()`
  - Will revisit this in second half of course
- Sets have similar **aliasing issues** as lists
- We can also mutate sets by using `+=`, `-=`, etc. because Python calls mutator methods when we use these operators
  - `s1 |= s2`, `s1 &= s2`, `s1 -= s2` are versions of `|`, `&`, `-` that mutate `s1` to become the result of the operation on the two sets.

Takeaways: Sets

- **Sets** are a new ***mutable*** unordered collection of immutable objects:
  - useful for eliminating duplicates from a collection if we don't care about losing order
  - can iterate over sets in a for loop (order will be arbitrary)
  - efficient way to store unordered objects when main application is checking membership `in` the set
  - can perform mathematical operations such as union, intersection, difference etc

Example in Class:

Using set to implement `get_candidates()`

Example in Class:

Using tuples to solve Madlibs Puzzles