2.2

Sets

Lesson Plan

- [10] Icebreaker
- [5] Set ADT
- [10] Sets in Python
- [10] Dealing with Duplicates
- [5] Hashable Types
- [20] Practice: Dogs!
- [30] Practice: Flight Itinerary

Icebreaker

Set ADT

Set ADT

A **Set** stores **unique values** in **no particular order**.

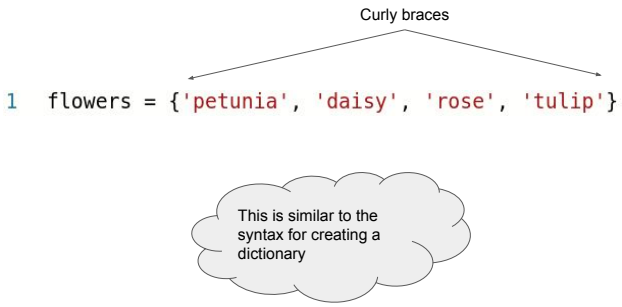
Supported Operations:

- **Insert** – add an element to a set if it is not already present
- **Lookup** – check if an element is in a set
- **Remove** – remove an element from a set

Sets in Python

Python tutorial, Python docs

Creating a Set



Creating an empty set

```
1  flowers = set()
```

YES

```
1  flowers = {}
```

NO

This will create an empty dictionary instead!

Sets are **unordered**!

```
1  flowers = {'petunia', 'daisy', 'rose', 'tulip'}
2  print(flowers)
```

{'daisy', 'tulip', 'rose', 'petunia'}

Looping through a set

```
1  flowers = {'petunia', 'daisy', 'rose', 'tulip'}
2
3  for flower in flowers:
4    print(flower)
```

Sets are unordered!

rose
tulip
petunia
daisy

Sets are **unordered**!

```
1  flowers = {'petunia', 'daisy', 'rose', 'tulip'}
2
3  for i in range(len(flowers)):
4    print(flowers[i])
```

Traceback (most recent call last):
File "main.py", line 4, in <module>
print(flowers[i])
TypeError: 'set' object is not subscriptable

Cardinality (# of items in the set)

```
1  flowers = {'petunia', 'daisy', 'rose', 'tulip'}
2  cardinality = len(flowers)
3  print(cardinality)
```

4

Check if a set is empty

Meh.

```
flowers = set()

if len(flowers) == 0:
  print('no flowers here!')
```

Style tip: do it the 'Pythonic' way!

```
flowers = set()

if not flowers:
  print('no flowers here!')
```

Lookup an item

```
1  flowers = {'petunia', 'daisy', 'rose', 'tulip'}
2
3  if 'daisy' in flowers:
4    print('flower!')
5  else:
6    print('not a flower!')
```

Lookup an item

```
1  flowers = {'petunia', 'daisy', 'rose', 'tulip'}
2
3  if 'mango' not in flowers:
4    print('not a flower')
```

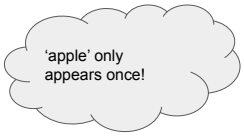
Insert an item

```
fruits = {'apple', 'banana'}
fruits.add('kiwi')
print(fruits)
```

{'kiwi', 'apple', 'banana'}

Items in a set are **distinct**

```
fruits = {'apple', 'banana'}
fruits.add('apple')
print(fruits)
```



{'banana', 'apple'}

Remove an item

```
fruits = {'apple', 'banana'}
fruits.remove('apple')
print(fruits)
```

{'banana'}

Removing an item that isn't there

```
fruits = {'apple', 'banana'}
fruits.remove('kiwi')
print(fruits)
```

```
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    fruits.remove('kiwi')
KeyError: 'kiwi'
```

Remove all items using clear()

```
1  flowers = {'petunia', 'daisy', 'rose', 'tulip'}
2
3  flowers.clear()
4  print(flowers)
```

```
set()
```

Dealing with Duplicates

Creating a set from a string

```
1  vowels = set('aeiou')
2  print(vowels)
```

Why might this be useful?

```
{'i', 'o', 'u', 'a', 'e'}
```

Creating a set from a list

The elements in a set are **unique**

```
1  snacks = ['cheetos', 'gushers', 'kitkats', 'cheetos', 'cheetos', 'kitkats', 'skittles']
2  distinct_snacks = set(snacks)
3  print(distinct_snacks)
```

```
{'skittles', 'kitkats', 'gushers', 'cheetos'}
```

Check for duplicates

How could I check if a list contains any duplicates?

```
def contains_duplicates(lst):
  return len(set(lst)) < len(lst)
```

If the set is smaller, the list must have had duplicates!

Removing duplicates from a list

This only works if the order of the result doesn't matter!

NON-MUTATING

```
def remove_duplicates(lst):
  return list(set(lst))
```

You can use the list() function to create a list from any sequence type

MUTATING

```
def remove_duplicates(lst):
  unique_items = list(set(lst))
  lst.clear()
  lst.extend(unique_items)
```

How could we change this function to modify the list in-place instead of returning a new one?

Practice: Remove duplicates **preserving order**

Write a function that removes duplicates from a list.

The order of the elements should be preserved in the result.

In other words, we want to keep the **first occurrence** of each element.

Example: **remove_duplicates**([3, 1, 1, 3, 2, 1]) should return [3, 1, 2]

Let's do this one together as a class!

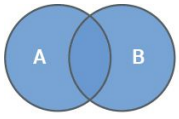
Remove duplicates preserving order

```
def remove_duplicates(lst):
  seen = set()
  unique_items = []
  for item in lst:
    if item not in seen:
      unique_items.append(item)
      seen.add(item)
  return unique_items
```

```
# In-place
def remove_duplicates(lst):
  seen = set()
  unique_items = []
  for item in lst:
    if item not in seen:
      unique_items.append(item)
      seen.add(item)
  lst.clear()
  lst.extend(unique_items)
```

Set Operations

Union

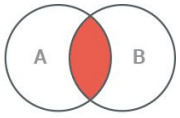


Union

```
1 cartoons = {'spongebob', 'scooby-doo', 'garfield', 'squidward'}
2 pets = {'hedwig', 'scooby-doo', 'garfield', 'lassie'}
3
4 characters = cartoons | pets
5
6 print(characters)
```

{'spongebob', 'scooby-doo', 'garfield', 'hedwig', 'lassie', 'squidward'}

Intersection



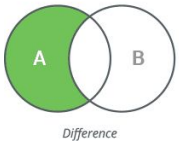
Intersection

```
1 cartoons = {'spongebob', 'scooby-doo', 'garfield', 'squidward'}
2 pets = {'hedwig', 'scooby-doo', 'garfield', 'lassie'}
3
4 characters = cartoons & pets
5
6 print(characters)
```

What does this print?

{'scooby-doo', 'garfield'}

Difference



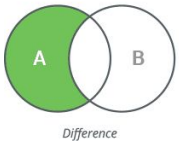
Difference

```
1 cartoons = {'spongebob', 'scooby-doo', 'garfield', 'squidward'}
2 pets = {'hedwig', 'scooby-doo', 'garfield', 'lassie'}
3
4 characters = cartoons - pets
5
6 print(characters)
```

What does this print?

```
{'spongebob', 'squidward'}
```

Difference (other direction)



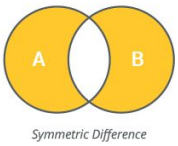
Difference

```
1 cartoons = {'spongebob', 'scooby-doo', 'garfield', 'squidward'}
2 pets = {'hedwig', 'scooby-doo', 'garfield', 'lassie'}
3
4 characters = pets - cartoons
5
6 print(characters)
```

What does this print?

```
{'lassie', 'hedwig'}
```

Symmetric Difference



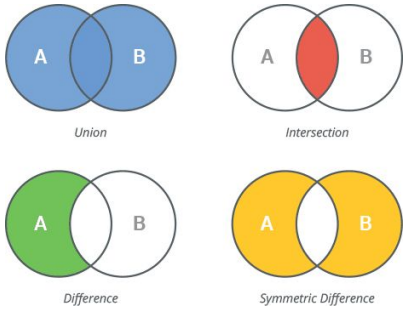
Symmetric Difference

```
1 cartoons = {'spongebob', 'scooby-doo', 'garfield', 'squidward'}
2 pets = {'hedwig', 'scooby-doo', 'garfield', 'lassie'}
3
4 characters = pets ^ cartoons
5
6 print(characters)
```

```
{'spongebob', 'lassie', 'squidward', 'hedwig'}
```

Summary

Union	A | B
Intersection	A & B
Difference	A - B
Symmetric Difference	A ^ B



Chat Waterfall: Perfect Date

You're trying to find a **bar** or **restaurant** that is **nearby** and not **expensive**.

Given the following sets:

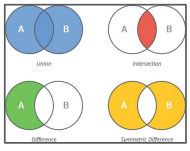
- **bars**
- **restaurants**
- **nearby**
- **expensive**

Construct a set containing the best options.

Chat waterfall:

To start with: What if you're just looking for a **bar or restaurant**?

Union	A | B
Intersection	A & B
Difference	A - B
Symmetric Difference	A ^ B



Perfect Date - Sample Solution

```
def perfect_date(bars, restaurants, nearby, expensive):
  return (bars | restaurants) & nearby - expensive
```

How do Dictionaries and Sets work?

Hashable Types

An object is *hashable* if it has a hash value which **never changes** during its lifetime.

We can only use hashable types as keys in dictionaries.

Some examples of hashable types are:

- String
- Integer
- Float
- Boolean

Are these types hashable?

- List of Integers e.g. [1, 2, 3]
  - No
- Tuple of Integers e.g. (1, 2, 3)
  - **Yes!**
- Tuple containing a List e.g. (1, [2, 3])
  - No
- Dictionary e.g. {1: 2, 3: 4}
  - No

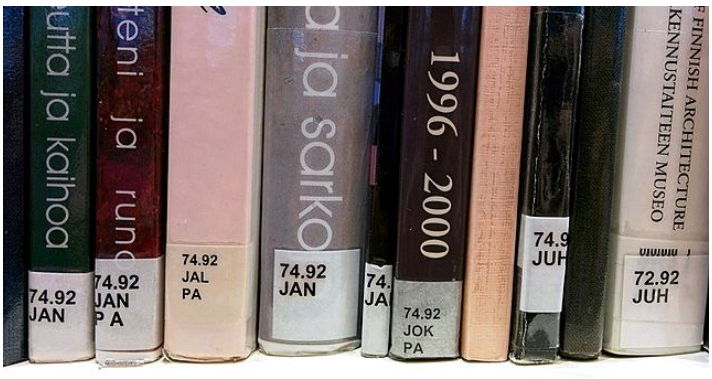
Immutable containers are hashable if their elements are hashable

Mutable containers are not hashable

Hashing

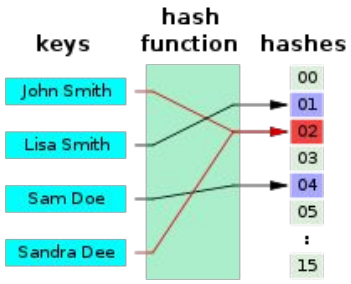
```
String: hello world!
Hash: -7505400730570130826

String: Tech Exchange!
Hash: -6252328129304768399

Int: 52
Hash: 52

Float: 52.1
Hash: 230584300921372724
```

Compute a unique* hash value for a given object

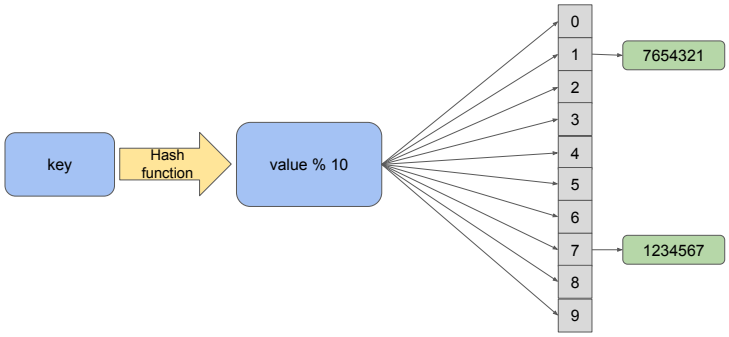


Hashing as a “Fingerprint”

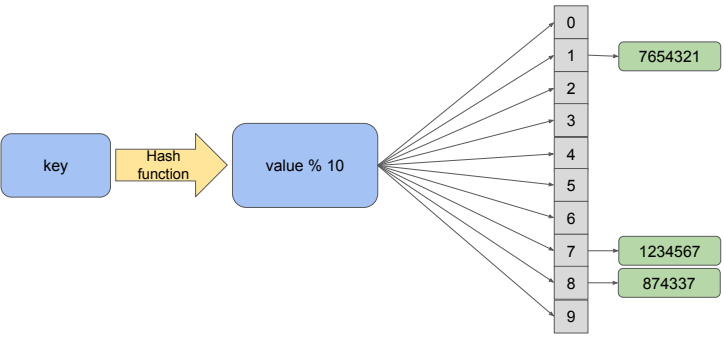
Hash Tables



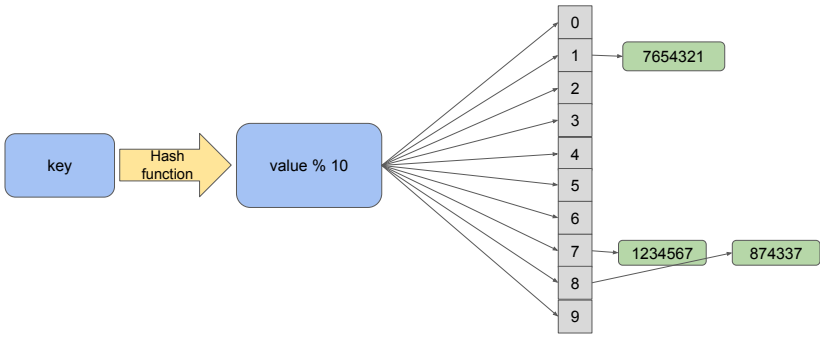
Example



What happens when there's a collision?



What happens when there's a collision?



A Good Hash Function

- Consistent
- Fast to compute
- Minimizes collisions

Practice Problem: Who let the dogs out?

You will be working in teams of 2 or 3. The goal is to collaboratively find a solution and be able to explain it to the class. Use the table below to figure out what your role is.

Role	Responsibilities	Assignment Criteria
Driver	Copy and share the repl.it, write the code, make sure you're listening to ideas from your teammates	Person with the most different colors in their outfit today
Tester	Play devil's advocate, thinks of edge cases, write unit tests for the driver's code	Person with the fewest different colors in their outfit today
Presenter	Document the code, be prepared to present the team's design decisions, and share one thing the team learned from the problem	Person with the middlest different colors in their outfit

If there are only 2 members in your team, the tester will also take on the presenter role.

```python
def possible(events):
  house = set()
  for dog, action in events:
    if action == 'in':
      if dog in house:
        return False
      else:
        house.add(dog)
    if action == 'out':
      if dog not in house:
        return False
      else:
        house.remove(dog)
  return True
```

Practice Problem: Flight Itinerary

You will be working in teams of 2 or 3. The goal is to collaboratively find a solution and be able to explain it to the class. Use the table below to figure out what your role is.

Role	Responsibilities	Assignment Criteria
Driver	Copy and share the repl.it, write the code, make sure you're listening to ideas from your teammates	Person with the most different colors in their outfit today
Tester	Play devil's advocate, thinks of edge cases, write unit tests for the driver's code	Person with the fewest different colors in their outfit today
Presenter	Document the code, be prepared to present the team's design decisions, and share one thing the team learned from the problem	Person with the middlest different colors in their outfit

If there are only 2 members in your team, the tester will also take on the presenter role.

Flight Itinerary - Sample Solution

```python
def itinerary(flights):
  flight_graph = {}
  for src, dst in flights:
    if src in flight_graph:
      raise Exception(f'More than one flight from {src}!')
    flight_graph[src] = dst
  # Use set for efficient lookup
  destinations = set(flight_graph.values())
  start = None
  for src in flight_graph:
    if src not in destinations:
      start = src
      break
  if start == None:
    raise Exception('Cycle found!')
  route = [start]
  current = start
  while current in flight_graph:
    current = flight_graph[current]
    route.append(current)
  return route
```