Stacks, Queues, & Sets

Basic Data Structure Review

- Lists
- Tuples
- Dictionaries

Programmers like abstractions

Lists and dictionaries are sometimes too general. By adding restrictions, programmers communicate how a given type should be used.

Stacks

Stack vocabulary

"Push" new elements on top of the stack

Remove the top by "popping" it

You can "peek" at the top element without removing it.

In other words, with a stack, you only ever manipulate the top element.

Stacks are LIFO

LIFO stands for:
"Last In, First Out"

What can a stack model?

- Python function calls and environments
- Task system with interruptions
- Tower of Hanoi game
- Dialog boxes

Stacks using Python lists

```
x = [5, 6, 9]
```

Pop: `previous_top = x.pop()`

Push: `x.append('32')`

Peek: `top = x[-1]`

Queues

How to use a queue?

Enqueue new elements to one end
Dequeue elements from the other

A queue is exactly like a line of people.

Queues are FIFO

FIFO stands for:
"First In, First Out"

When are queues useful?

- Coloring Scrabble squares
- Processing a set of requests fairly

Queues in Python via lists

```
x = ['a', 'b', 'c']
```

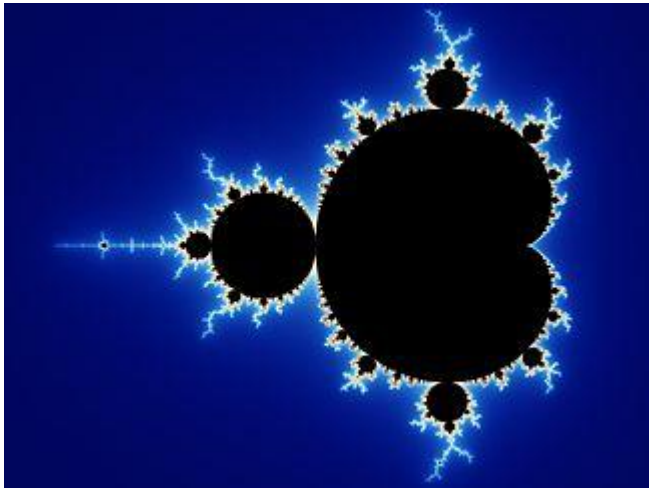
Dequeue: `previous_front = x.pop(0)`

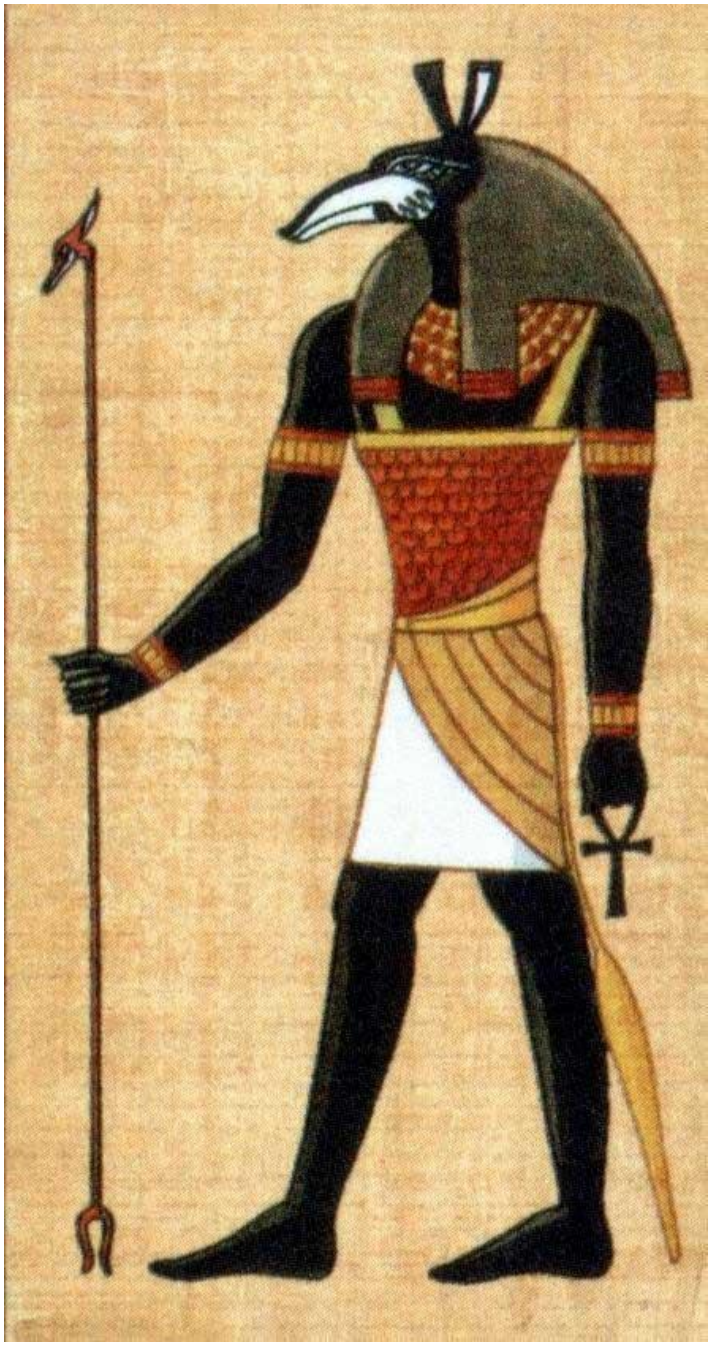
Enqueue: `x.append('d')`

Peek: `front = x[0]`

Sets


OVER
25
BEST
GAME
AWARDS
SET
THE FAMILY GAME
OF VISUAL PERCEPTION
Instructions en Francais
Instrucciones en Español





Set definition

An unordered collection of unique elements.

Like a list, but each element can only appear once.

A set is unlike a list because it's not ordered. (Kind of like dictionary keys.)

It's a handy way to take an arbitrary collection of data but only process unique elements.

Sets in Python

```
>>> x = set([1, 1, 3, 2, 2, 2, 1, 0])
>>> print x
set([0, 1, 2, 3])
```

Sets in Python

```
>>> x = set([1, 1, 3, 2, 2, 2, 1, 0])
>>> print x
set([0, 1, 2, 3])

>>> 3 in x
True
```

Sets in Python

```
>>> x = set([1, 1, 3, 2, 2, 2, 1, 0])
>>> print x
set([0, 1, 2, 3])

>>> 3 in x
True

>>> len(x)
4
```

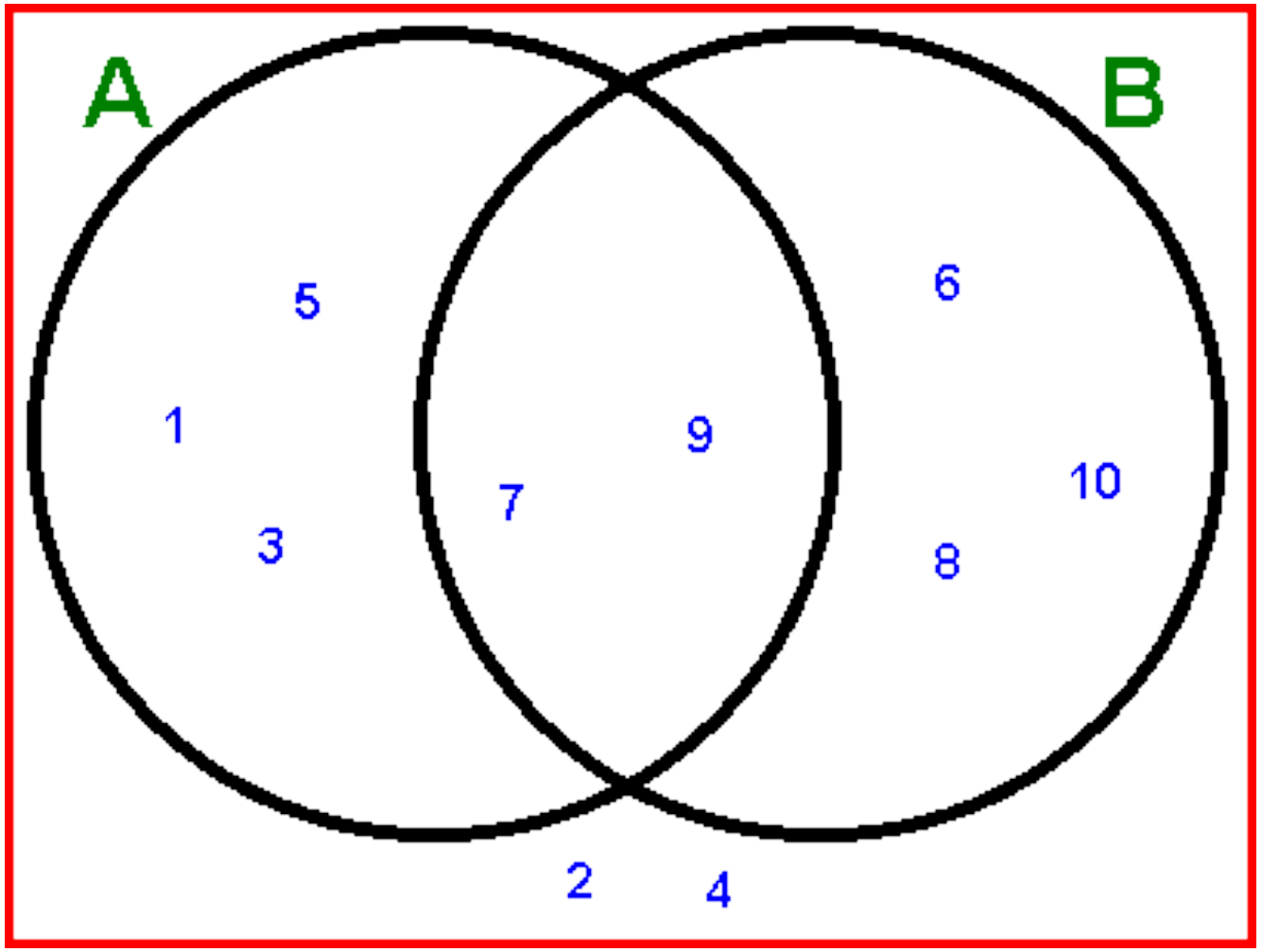
More details

Sets are initialized with set(some_list). You can think of this as "converting a list into a set", like you convert a string to an integer with int ("3").

You can loop over sets with a for loop like a list.

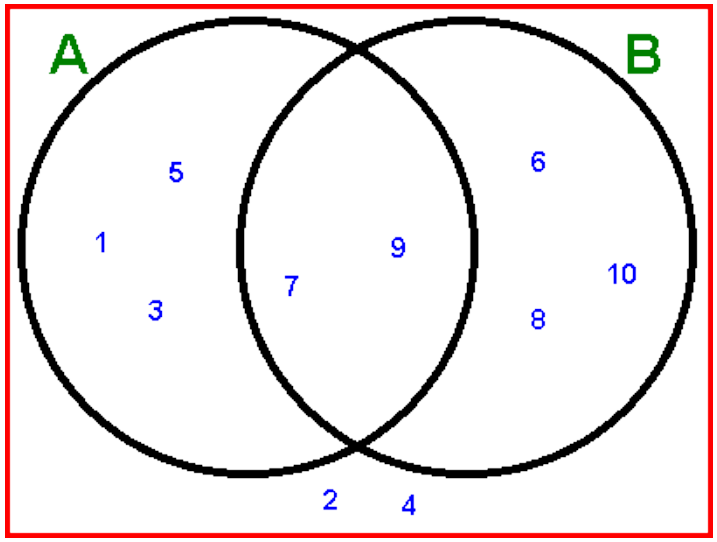
Sets are unordered, so you cannot do a[0] to get the first element of the set a.

Venn Diagrams



...in Python

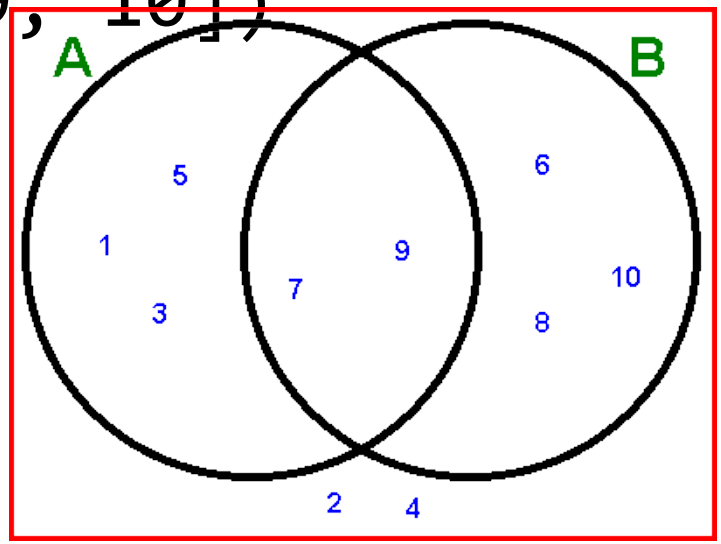
```
>>> a = set([1, 3, 5, 7, 9])
>>> b = set([6, 7, 8, 9, 10])
```



...in Python

```
>>> a = set([1, 3, 5, 7, 9])
>>> b = set([6, 7, 8, 9, 10])

>>> a.union(b)
set([1, 3, 5, 6, 7, 8, 9, 10])
```

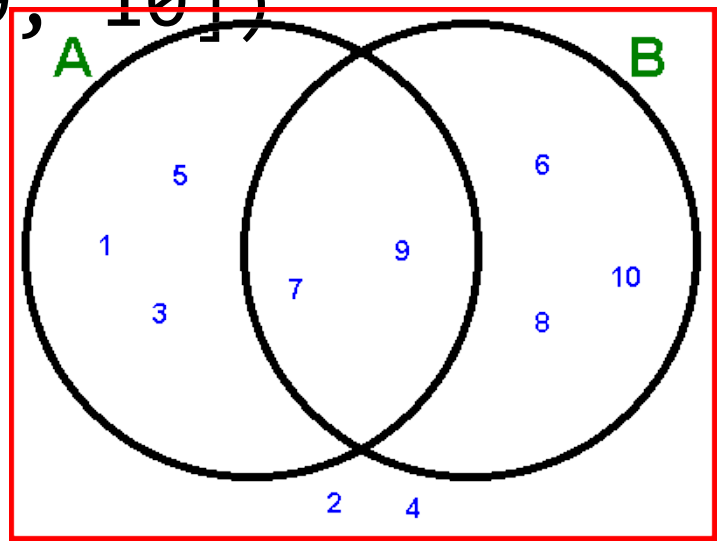


...in Python

```
>>> a = set([1, 3, 5, 7, 9])
>>> b = set([6, 7, 8, 9, 10])

>>> a.union(b)
set([1, 3, 5, 6, 7, 8, 9, 10])

>>> a.intersection(b)
set([7, 9])
```



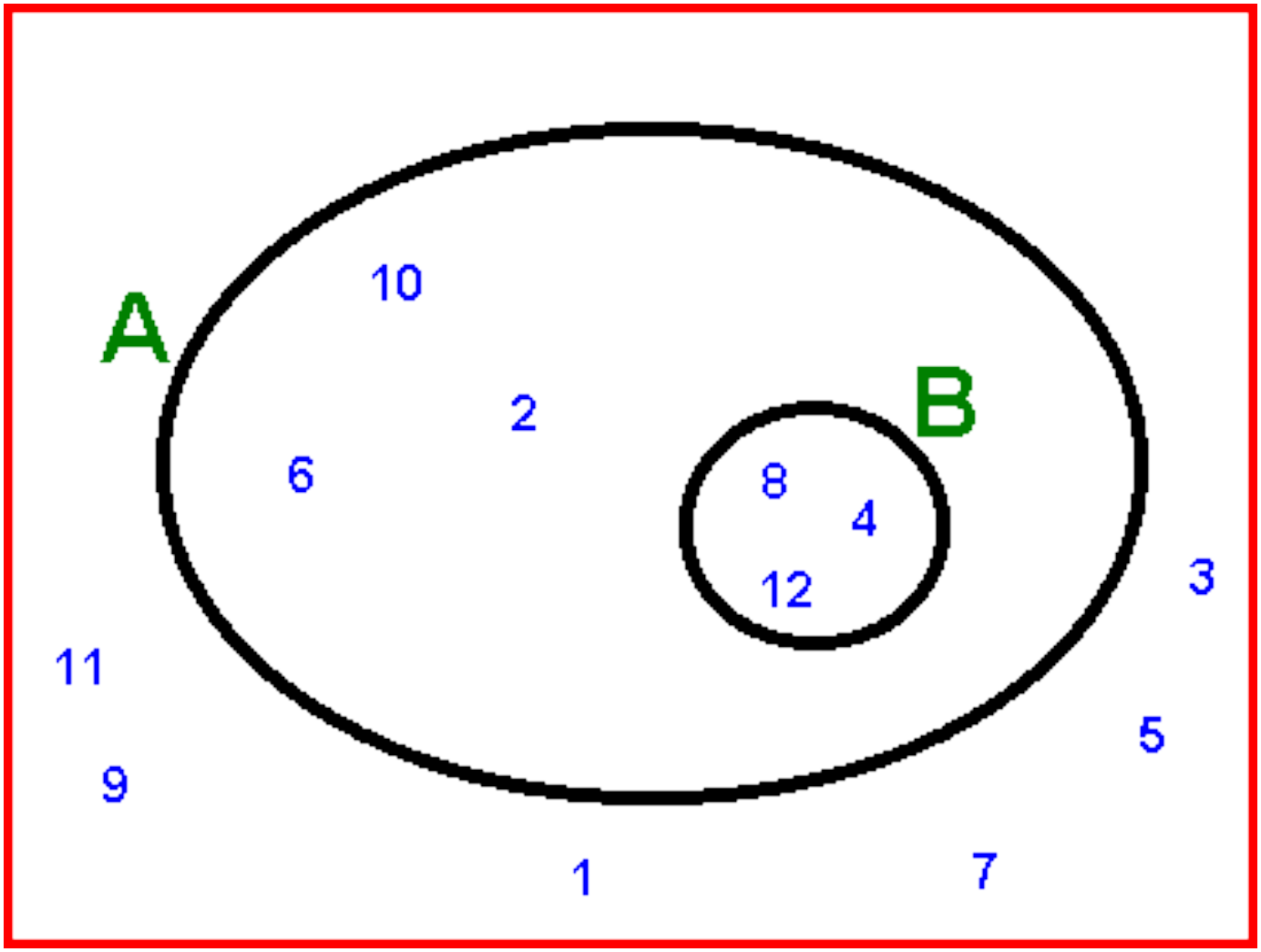
Python sets

Union and intersection functions return a new set and don't modify the original set.

In other words:

```
# WRONG; THIS CODE DOESN'T WORK
x = set([1, 2, 3])
y = set([4, 5])
x.union(y)
```

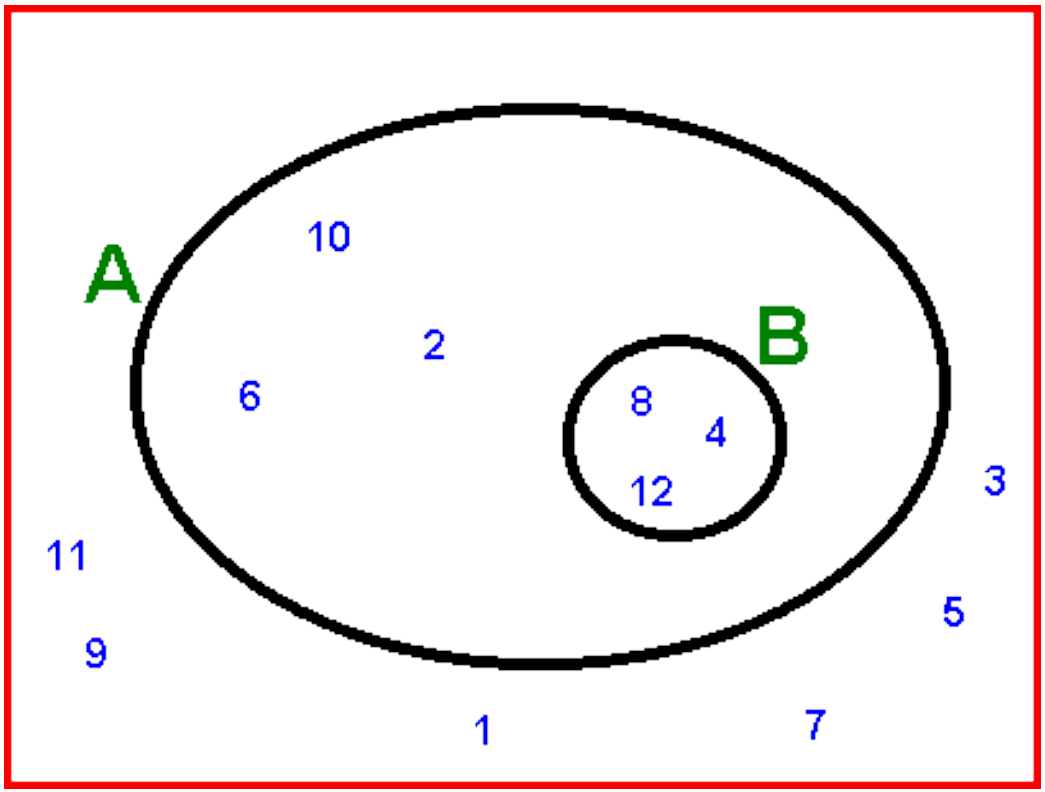
Venn Diagrams, part 2



Subsets / supersets

```
>>> a = set([2, 4, 6, 8, 10, 12])
>>> b = set([4, 8, 12])

>>> b.subset(a)
True
```

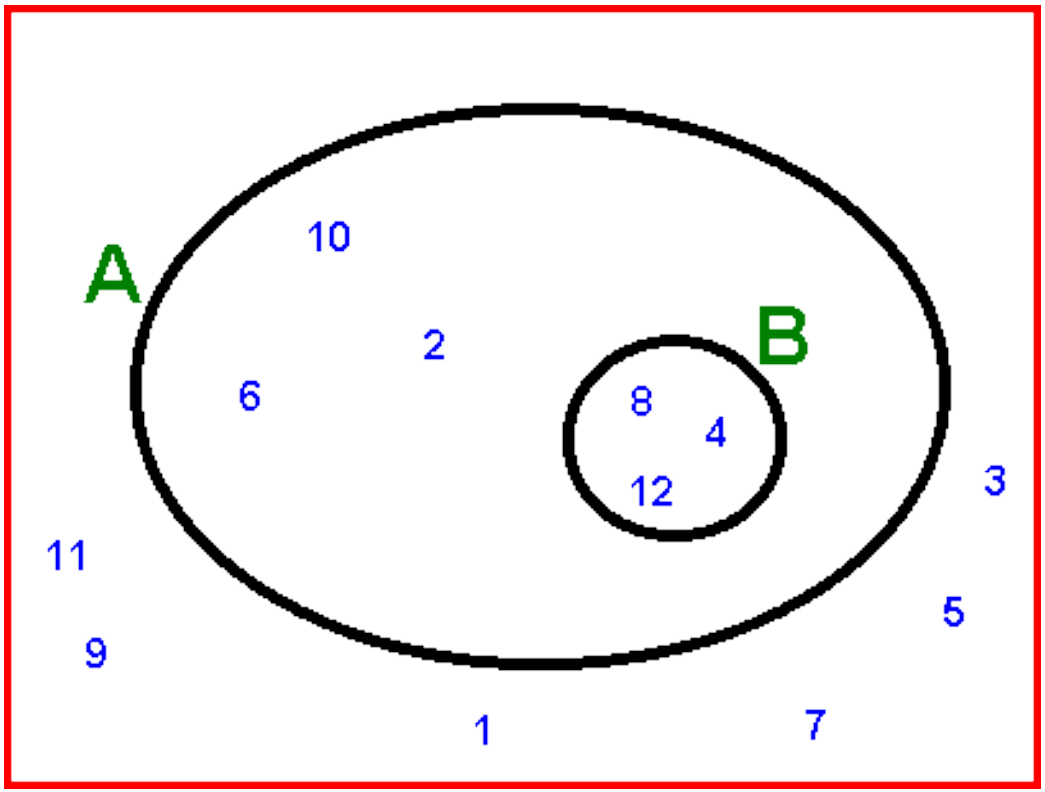


Subsets / supersets

```
>>> a = set([2, 4, 6, 8, 10, 12])
>>> b = set([4, 8, 12])

>>> b.subset(a)
True

>>> a.subset(b)
False
```



Dictionary implementation

Why oh why are dictionary keys unsorted??

What good is a dictionary?? What can I do with my_dict that I can't do with my_list?

```
my_dict = { 'a': 3, 'b': 4 }
my_list = [ ('a', 3), ('b', 4) ]
```

Dictionary = fast lookup

Organization, Python style

In the real world, what happens when you add another event?

Also, what happens when you have a bunch of events that all start with the same letter?

Python restrictions

Like a list, a dictionary is stored in memory with a fixed number of buckets. It's expensive to add more buckets.

So, you have some fixed number of buckets to store your values in, and need some way to map keys (which could be any type!) to bucket index.

How do you map hashes to indices?

If you have 8 buckets, how do you map a key to one of those buckets as quickly as possible?

```
>>> hash('a')
12416037344
```

"Hash" function

A hashing function is a function that takes a value of any type and returns an index for that value.

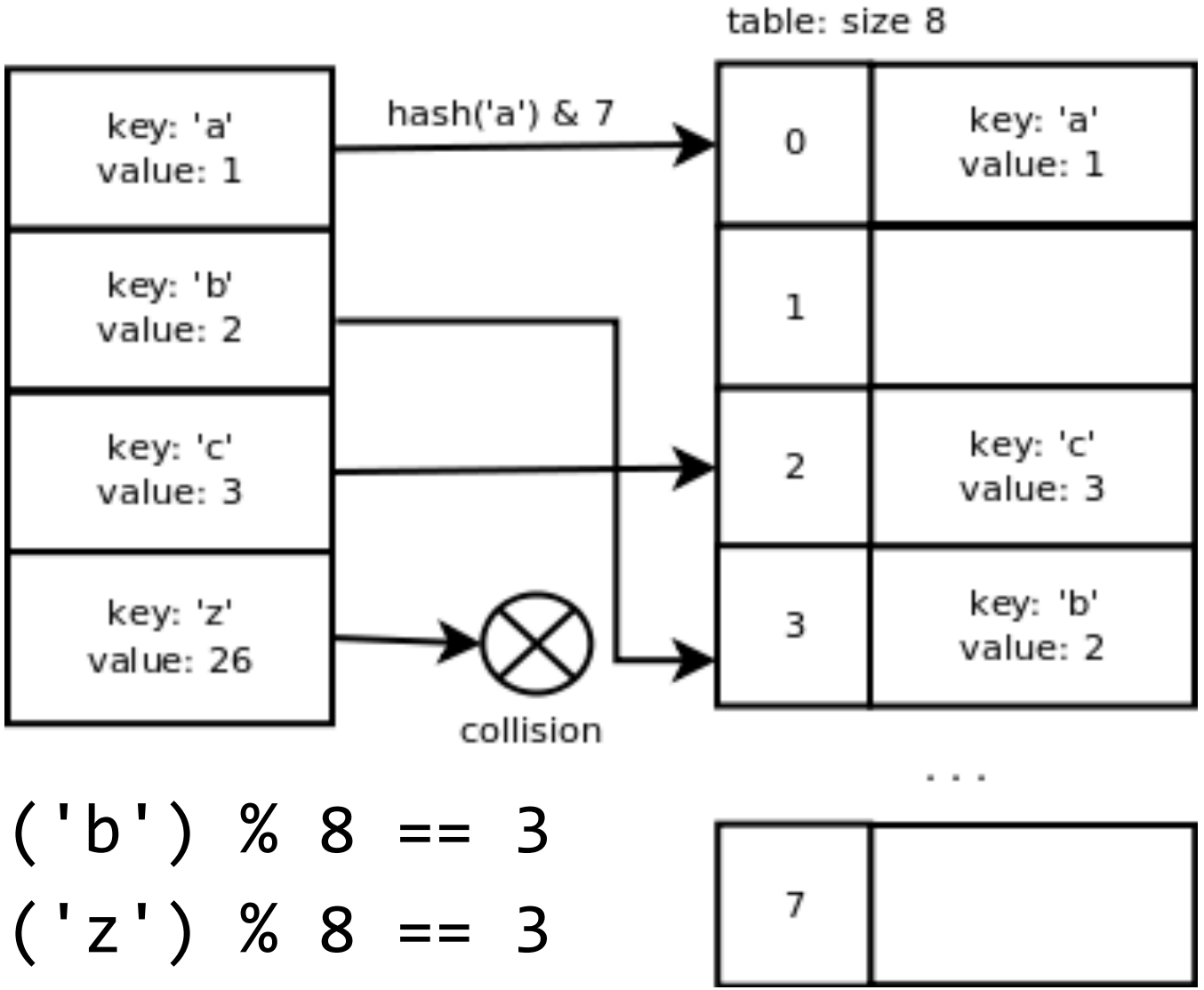
These are hard to write well.

How do you map hashes to indices?

Answer: Use the modulus operator!

```
>>> hash('a') % 8
0
>>> hash('b') % 8
3
>>> hash((4, 8)) % 8
7
```

Dictionaries, under the hood

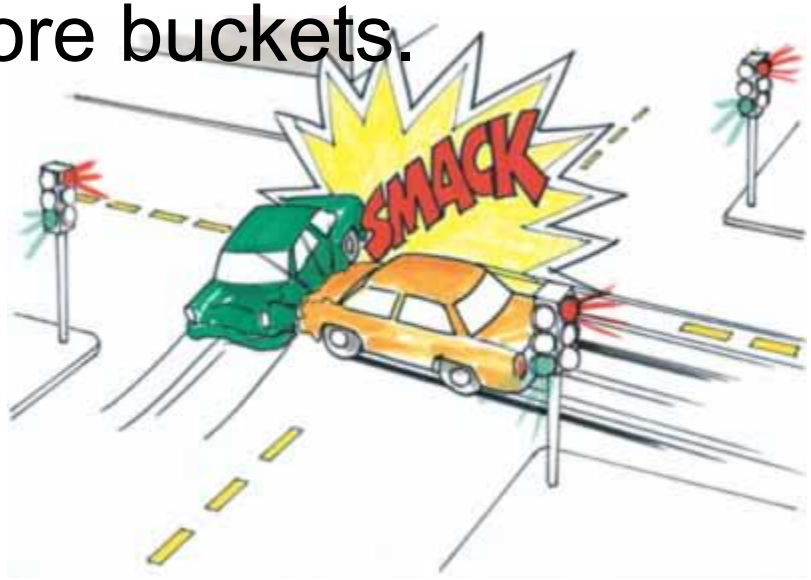



```
hash('b') % 8 == 3
hash('z') % 8 == 3
```

Collisions

It's called a collision when two keys "hash" to the same index.

One way to handle this is just to change the size of the list and add more buckets.



Collisions

...but oh no! If we're going from 8 buckets:

```
>>> hash('z') % 8
3
```

...and now we have 16 buckets:

```
>>> hash('z') % 16
11
```

Collision Resolution Option #2

If the bucket you want is full, then put it in some other bucket.

Unfortunately, this makes the "does this dictionary key contain this key" operation require looking at multiple buckets.

Dictionary = hash map

This is why dictionaries are often called "hash maps".