Data Structures part 2

CSCI 145

October 20, 2025

Outline

- **References**
- Sets
- More recursive drawing

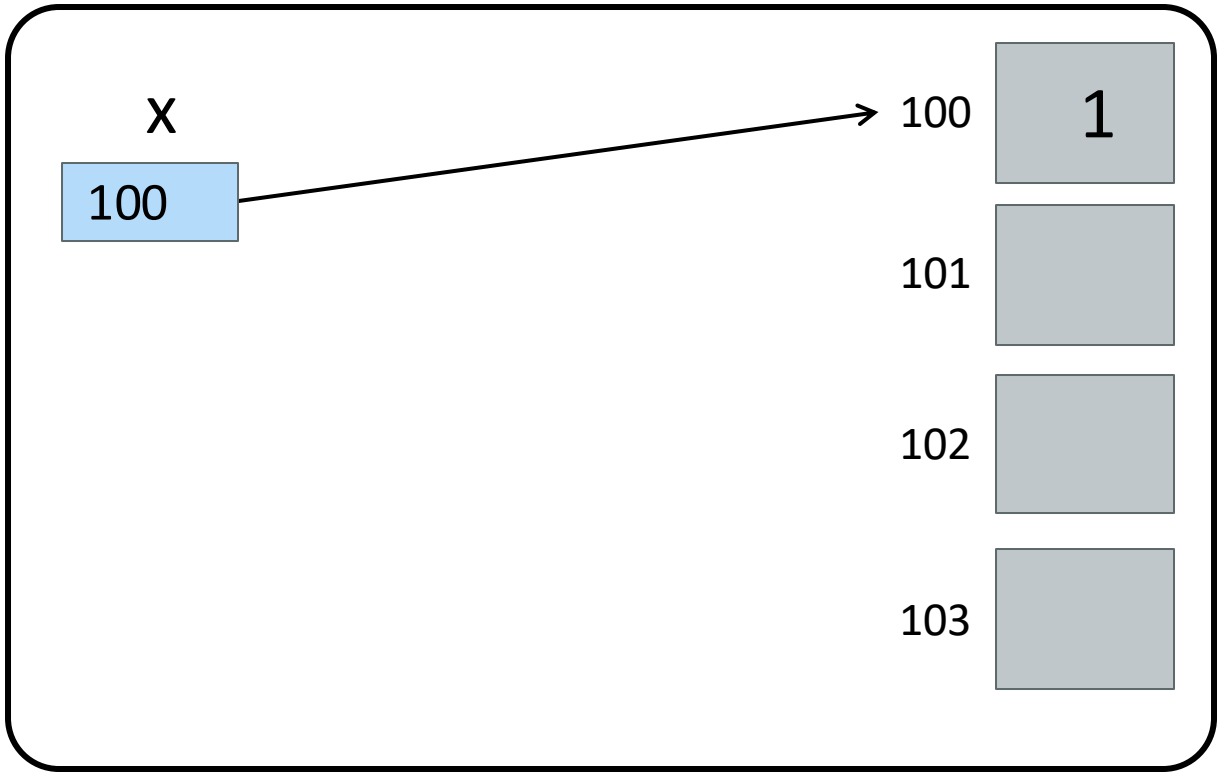
References

- Python variables *actually* store **references**, not **values**
- **References** are the **address in memory** where the data is found
  - You can find the address of a variable using the id function
  - The keyword is tells us if two objects are references to the same memory address
- This is only a critical distinction for **mutable** data structures (e.g., lists and dictionaries)
  - This is because we can alter the contents of a mutable data structure without using an assignment statement to change the contents of the variable

Memory Model

With basic data types (int/float/str/bool)

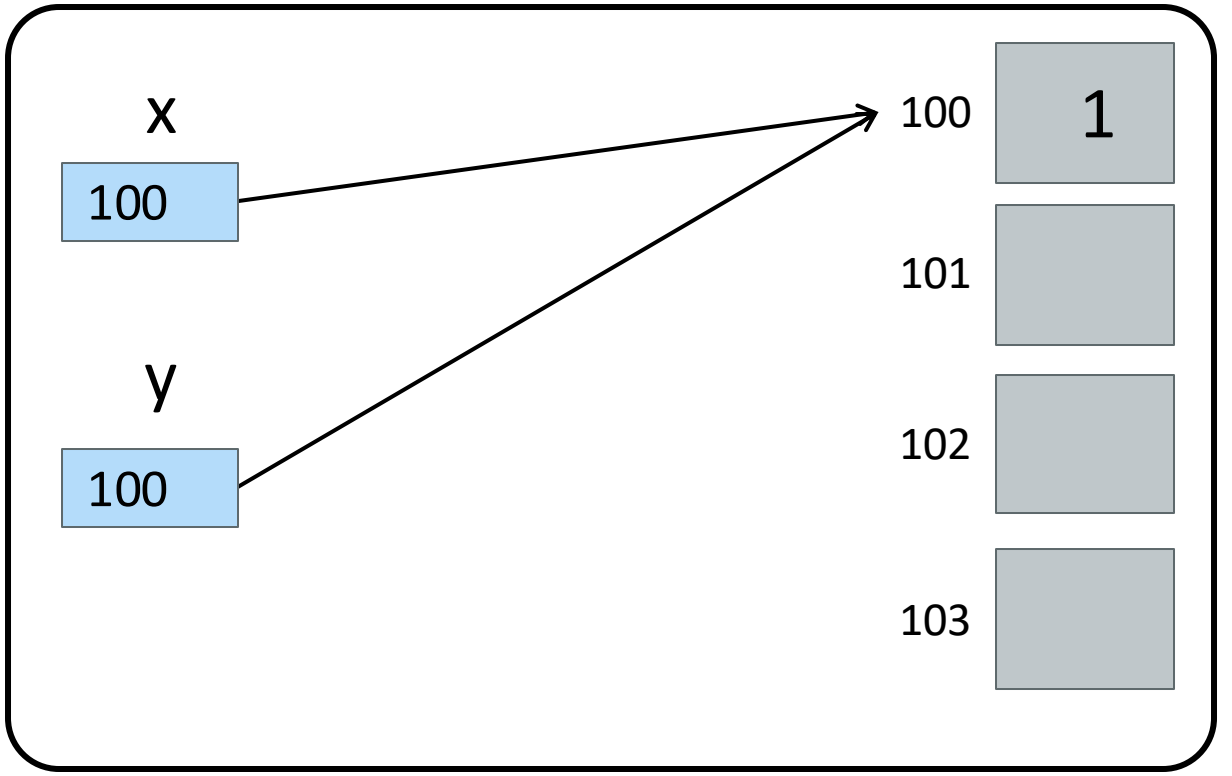
x = 1



Memory Model

With basic data types (int/float/str/bool)

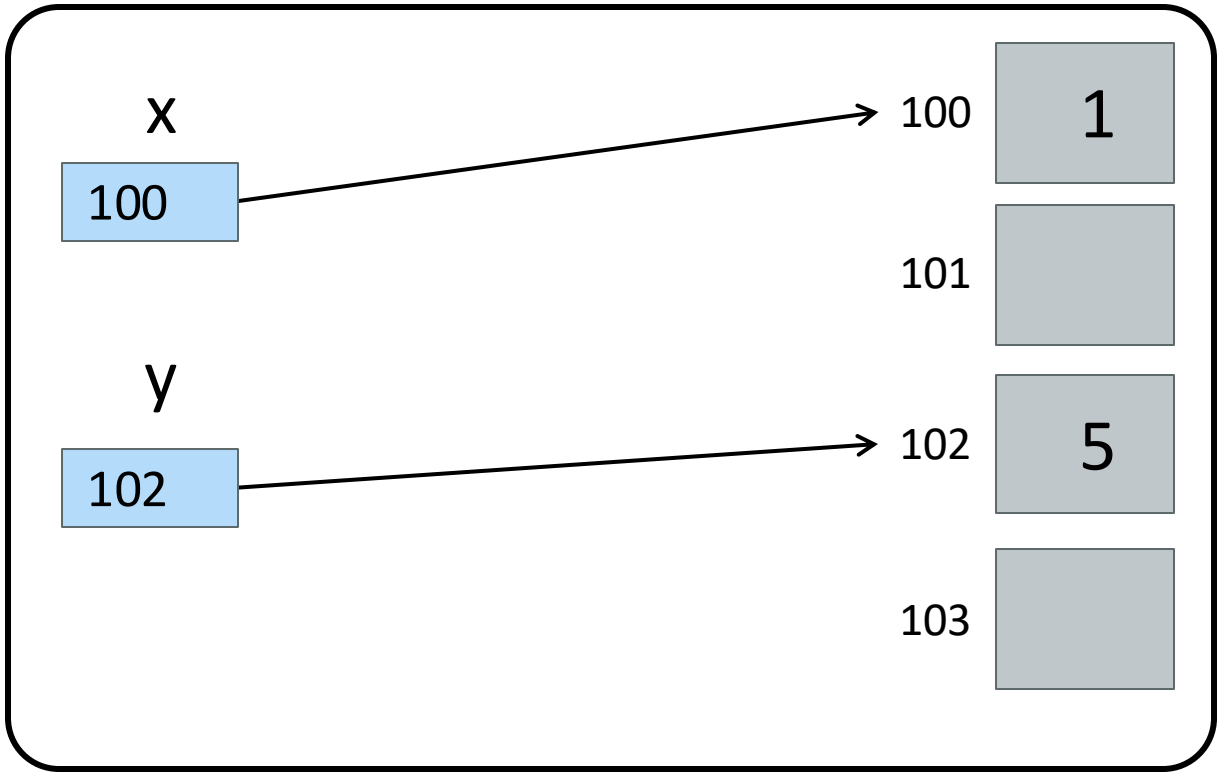
```
x = 1
y = x
```



Memory Model

With basic data types (int/float/str/bool)

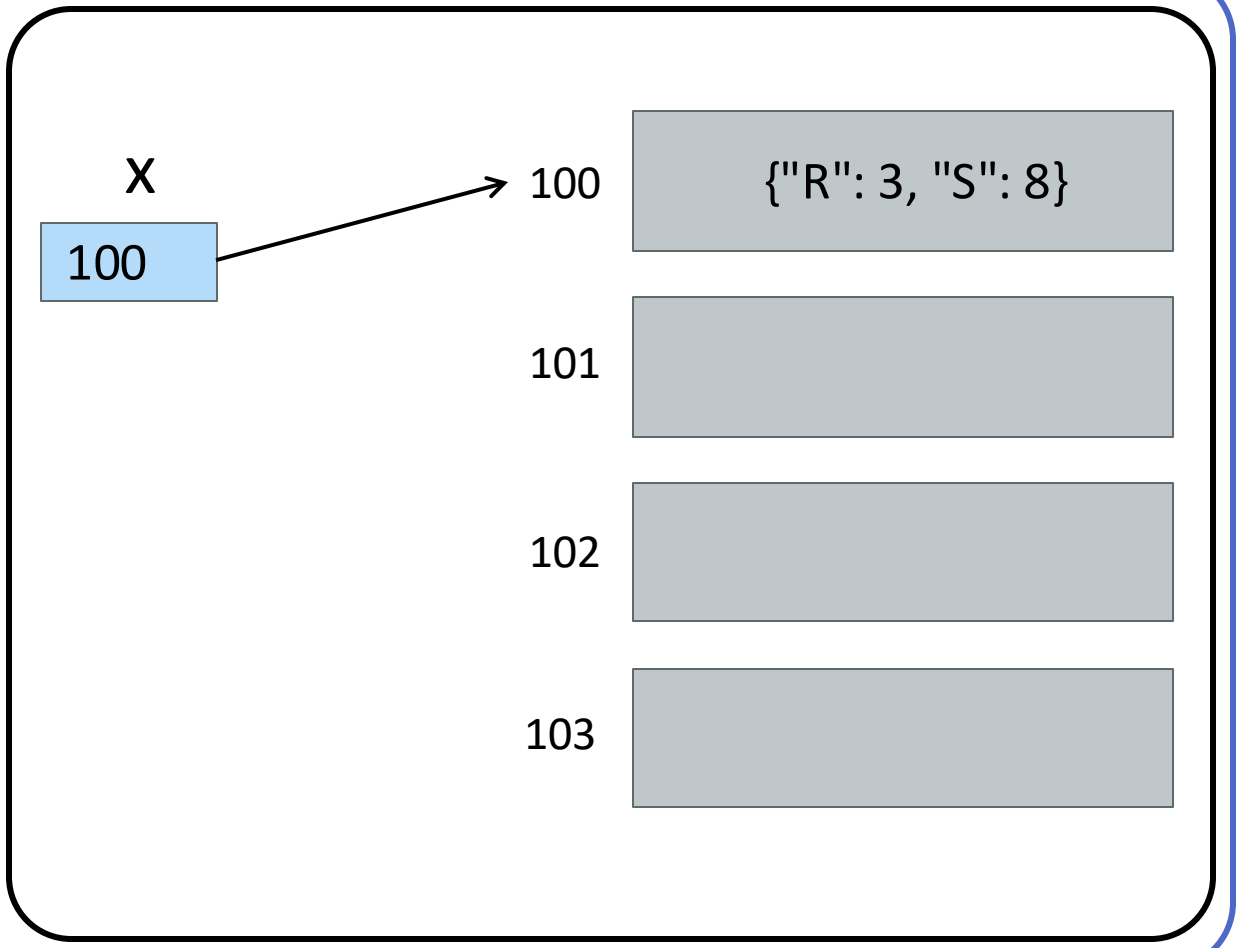
```
x = 1
y = x
y = 5
```



Memory Model

With mutable data types stored as references

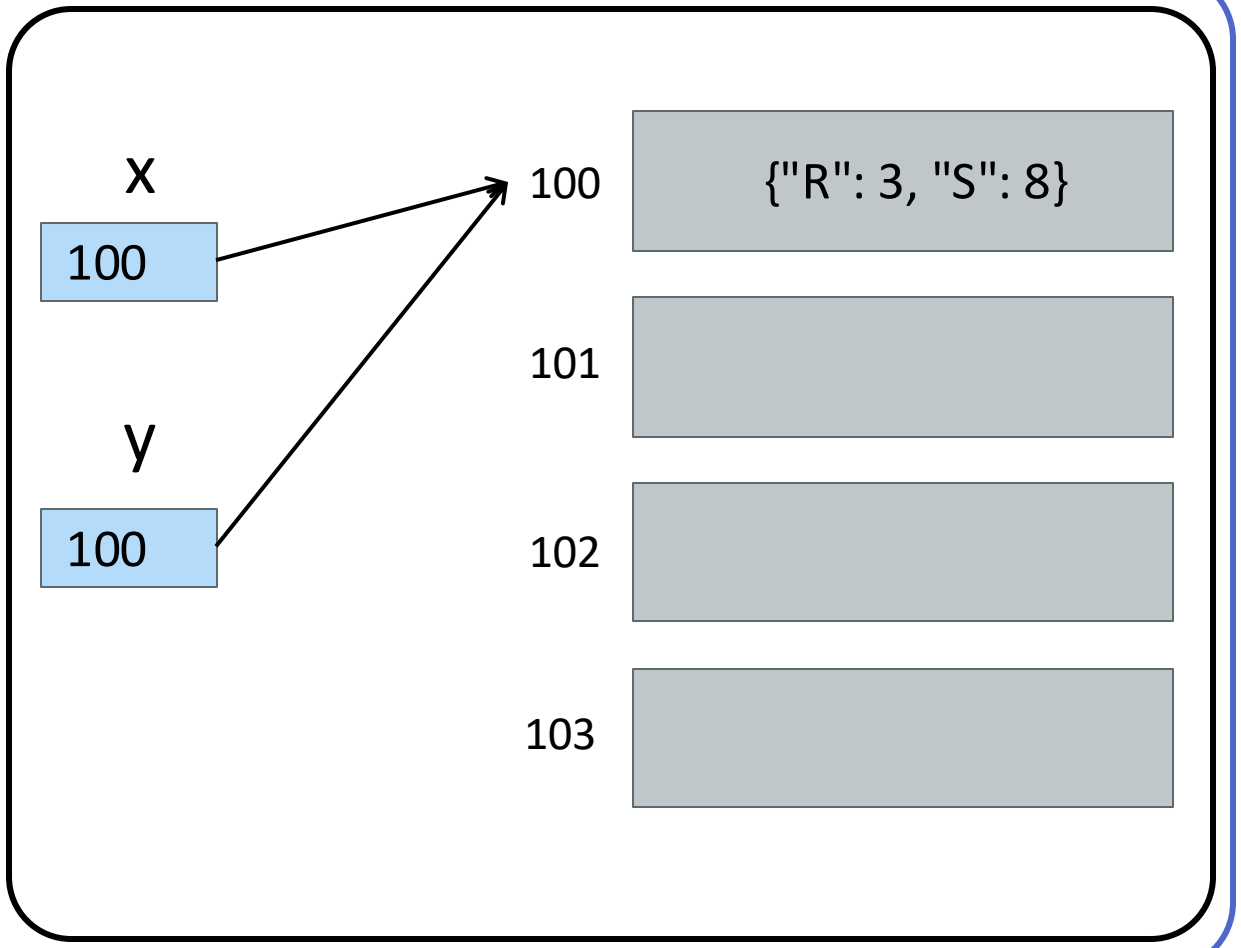
```
x = {"R": 3, "S": 8}
```



Memory Model

With mutable data types stored as references

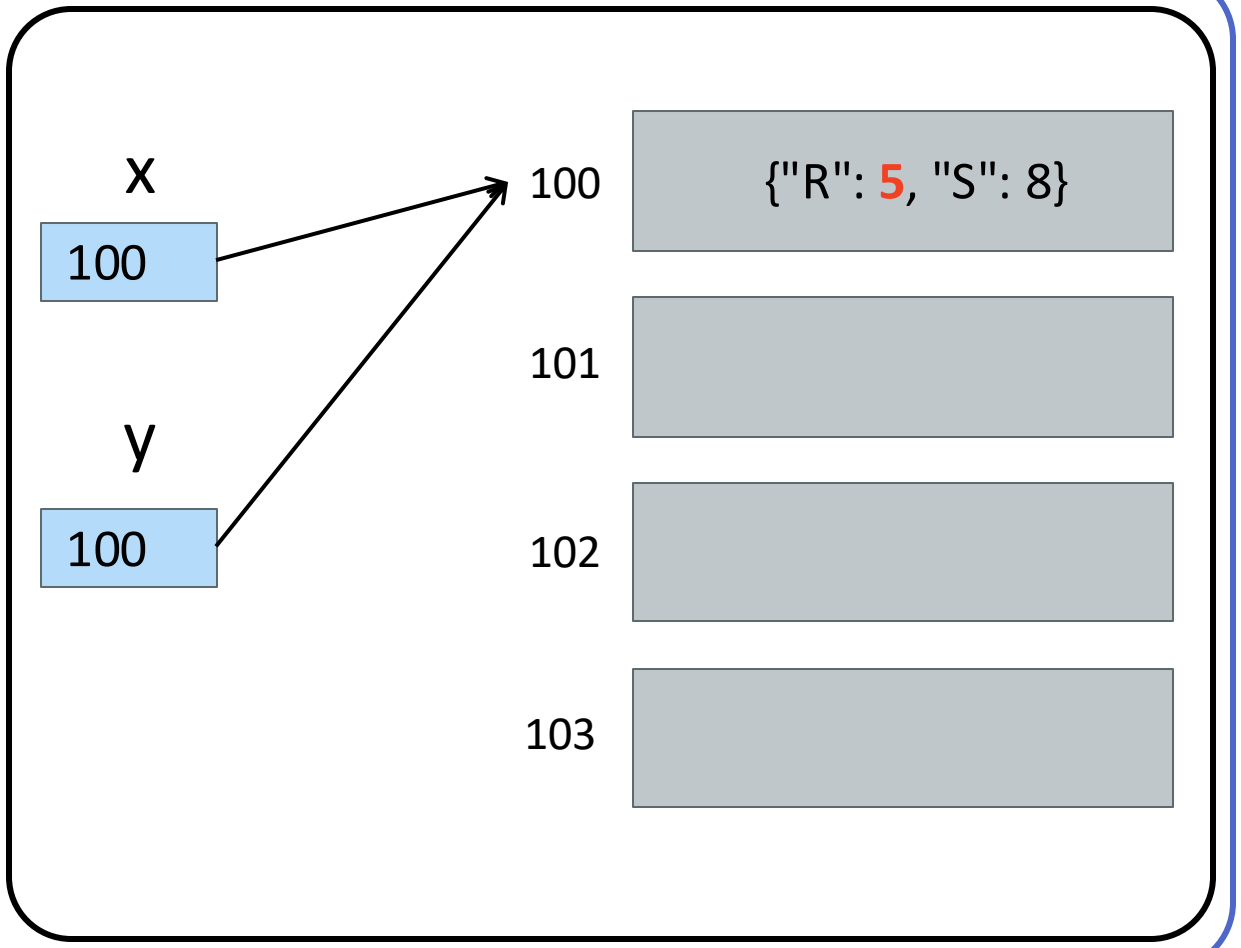
```
x = {"R": 3, "S": 8}
y = x
```



Memory Model

With mutable data types stored as references

```
x = {"R": 3, "S": 8}
y = x
y["R"] = 5
```



Copying

- Most mutable types provide a .copy() method that allows you to create a copy

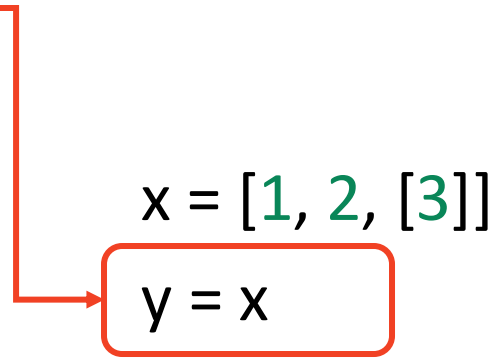
- Consider the following example. What you you expect y to be?

```
x = [1, 2, [3]]
y = x.copy()
x[-1][0] = 4
```

- Answer: [1, 2, [4]]

References: Deep vs. Shallow Copies

- Alias
  - Simply give another name to the same reference



```
x = [1, 2, [3]]
y = x
x[0] = 5
x[-1][0] = 4

print(x)
print(y)
```

References: Deep vs. Shallow Copies

- Alias
  - Simply give another name to the same reference
- Shallow Copy
  - Creates a new object, but keeps references from the original object

```
x = [1, 2, [3]]
y = x.copy() # x[:], list(x)
x[0] = 5
x[-1][0] = 4

print(x)
print(y)
```

References: Deep vs. Shallow Copies

- Alias
  - Simply give another name to the same reference
- Shallow Copy
  - Creates a new object, but keeps references from the original object
- Deep Copy
  - Creates a new object, and also **copies** references from the original object (recursively)

```
x = [1, 2, [3]]
y = copy.deepcopy(x)
x[0] = 5
x[-1][0] = 4

print(x)
print(y)
```

Exercise: References

Consider the following code:

```
x = [1, 3, 4]
y = x
y.append(4)
x[0] = 10
```

What is x after executing the code? What is y?

Outline

- References
- **Sets**
- More recursive drawing

Sets

- Sets are an **unordered collection** of items
- Each item may appear only once in a set
- Creating an empty set:
  - s = set()
  - Common error: don’t use {} (that is an empty dictionary)
- Creating a non-empty set:
  - s = {1, 2}

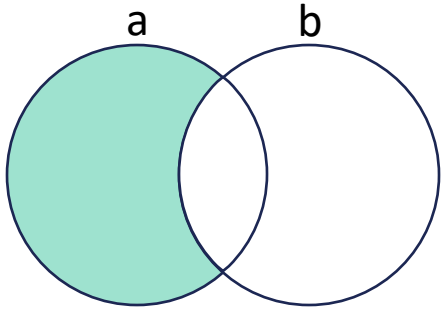
Set Methods

add

```
s = {3}
s.add(1)
print(s) # prints {1, 3}
```

What happens if you add something that's already in the set? **Nothing!**

Difference

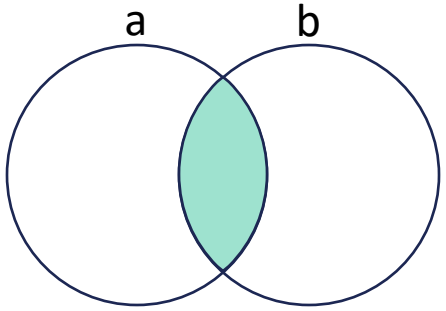


Python syntax:

- a.difference(b)
- a - b

All of the elements from set a that are not in set b

Intersection

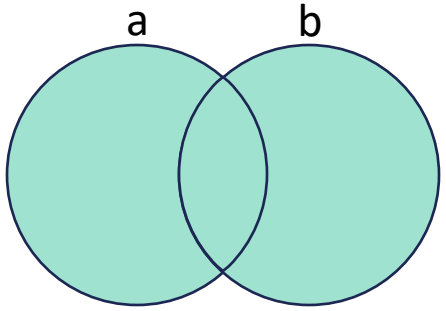


Python syntax:

- a.intersection(b)
- a & b

All of the elements that are in **both** set a and set b

Union



Python syntax:

- a.union(b)
- a | b

All of the elements that are in **either** set a or set b

Check if Something is in a Set

- Use the **in** operator, just like you would for a string, a list, or a dictionary
- One cool thing: this is faster with sets than it is with lists
  - Want to know how this is possible? Take CS 201!

Exercise: which data structure?

Which data structure (or structures) would you use to represent the following?

- Your ordered TODO list, from most to least important
- Each unique character in a string
- The heights of each mountain peak in Vermont
- A course roster with names, emails, and class years

Outline

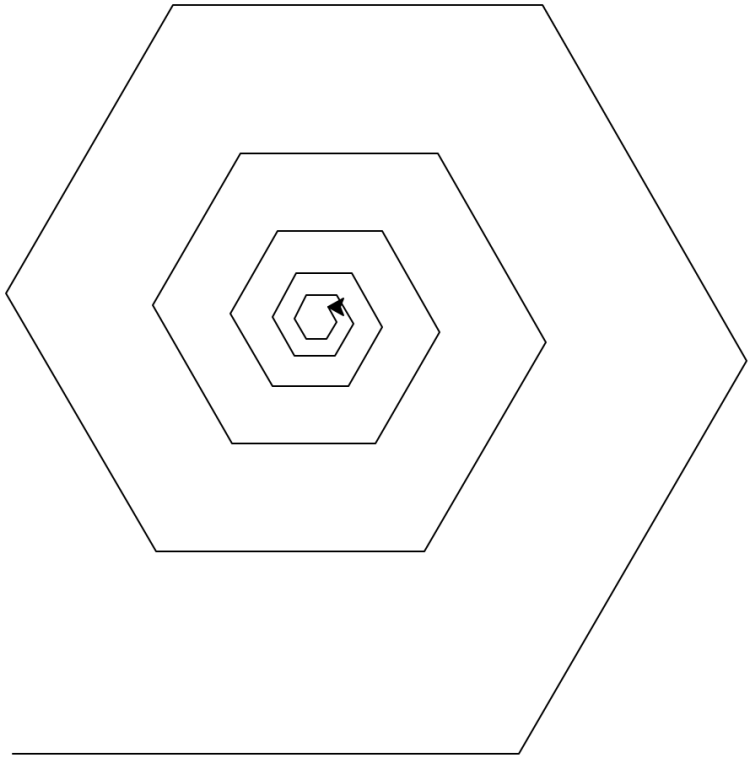
- References
- Sets
- **More recursive drawing**

Recursive drawing with the turtle - Spiral

```
import turtle as t


def spiral(length):
    if length > 10:
        t.forward(length)
        t.left(60)
        spiral(length * .9)


spiral(300)
```

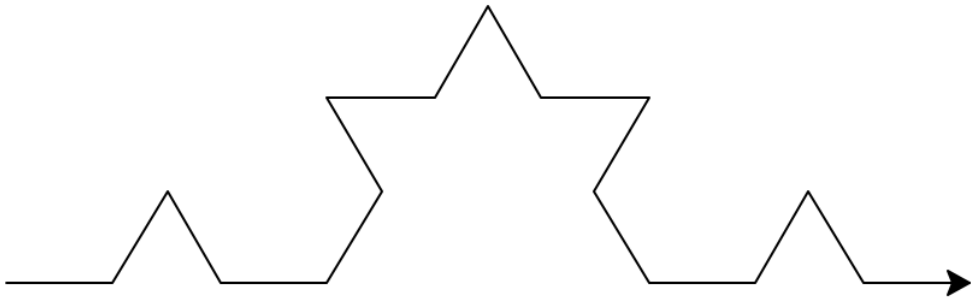


Recursive drawing with the turtle - Koch

```
import turtle as t

def draw_koch(length, generations):
 if generations == 0:
       t.forward(length)
    else:
       draw_koch(length/3, generations - 1)
       t.left(60)
       draw_koch(length/3, generations - 1)
       t.right(120)
       draw_koch(length/3, generations - 1)
       t.left(60)
       draw_koch(length/3, generations - 1)

draw_koch(300, 3)
```



Recursive drawing with the turtle - Tree

```
import turtle as t

def draw_tree(length, levels):
    if levels > 0:
        t.forward(length)
        t.left(45)
        draw_tree(length/2, levels-1)
        t.right(90)
        draw_tree(length/2, levels-1)
        t.left(45)
        t.backward(length)

draw_tree(200, 5)
```