

# Sets

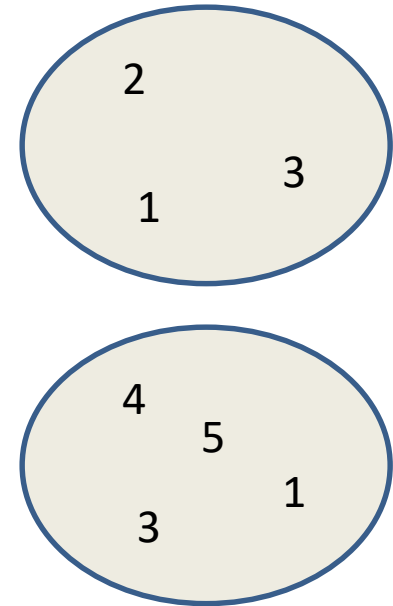
Ruth Anderson

UW CSE 160

Autumn 2021

# Sets

- Mathematical set: a collection of values, without duplicates or order
- Order does not matter  
 $\{ 1, 2, 3 \} == \{ 3, 2, 1 \}$
- No duplicates  
 $\{ 3, 1, 4, 1, 5 \} == \{ 5, 4, 3, 1 \}$
- For every data structure, ask:
  - How to create
  - How to query (look up) and perform other operations
    - (Can result in a new set, or in some other datatype)
  - How to modify



Answer: <http://docs.python.org/3/library/stdtypes.html#set>

# Two ways to create a set

1. Direct mathematical syntax:

```
odd = {1, 3, 5}
```

```
prime = {2, 3, 5}
```

Note: **Cannot use “{ }” to express empty set**: it means empty dictionary! Use **set()** instead.

2. Construct from a **list**: (also from a tuple or string)

```
odd = set([1, 3, 5])
```

```
prime = set([2, 3, 5])
```

```
empty = set([]) # or set()
```

# Set operations

```
odd = {1, 3, 5}
prime = {2, 3, 5}
```

- membership  $\in$       Python: `in`      `4 in prime`  $\Rightarrow$  False
- union  $\cup$       Python: `|`      `odd | prime`  $\Rightarrow$  {1, 2, 3, 5}
- intersection  $\cap$       Python: `&`      `odd & prime`  $\Rightarrow$  {3, 5}
- difference  $\setminus$  or  $-$       Python: `-`      `odd - prime`  $\Rightarrow$  {1}

Think in terms of set operations,  
*not* in terms of iteration and element operations  
– Shorter, clearer, less error-prone, faster

Although we can do iteration over sets:

```
# iterates over items in arbitrary order
for item in myset:
```

...

But we cannot index into a set to access a specific element.

# Practice with sets

```
z = {5, 6, 7, 8}
```

```
y = {1, 2, 3, 1, 5}
```

```
k = z & y
```

```
j = z | y
```

```
m = y - z
```

```
n = z - y
```

# Modifying a set

- **Add** one element to a set:

```
myset.add(newelt)  
myset = myset | {newelt}
```

- **Remove** one element from a set:

```
myset.remove(elt)    # elt must be in myset or raises error  
myset.discard(elt)   # never errors  
myset = myset - {elt}  
What would this do?  
myset = myset - elt
```

- Remove and return an arbitrary element from a set:  
`myset.pop()`

**Note: add, remove and discard all return None**

# Practice with sets

```
z = {5, 6, 7, 8}
y = {1, 2, 3, 1, 5}
p = z
q = set(z)    # Makes a copy of set z
z.add(9)
q = q | {35}
z.discard(7)
q = q - {6, 1, 8}
```

# Aside: List vs. set operations (1)

Find the common elements **in both** list1 and list2:

```
out1 = []  
for elem in list2:  
    if elem in list1:  
        out1.append(elem)
```

---

Find the common elements **in both** set1 and set2:

**set1 & set2**

Much shorter, clearer, easier to write with sets!



# Aside: List vs. set operations(2)

Find elements in **either** list1 or list2 (or both) (without duplicates):

```
out2 = list(list1)          # make a copy
for elem in list2:
    if elem not in list1: # don't append elements already in out2
        out2.append(elem)
```

Another way:

```
out2 = list1 + list2  # if an item is in BOTH lists, it will appear TWICE!
for elem in out1:      # out1 = common elements in both lists
    out2.remove(elem)  # Remove common elements, leaving just a single copy
```

---

Find the elements in **either** set1 or set2 (or both):

**set1 | set2**

# Aside: List vs. set operations(3)

Find the elements in **either list but not** in both:

```
out3 = []
out2 = list1 + list2  # if an item is in BOTH lists, it will appear TWICE!
for elem in out2:
    if elem not in list1 or elem not in list2:
        out3.append(elem)
```

---

Find the elements in **either set but not** in both:

**set1 ^ set2**

# Not every value may be placed in a set

- Set elements must be **immutable** values
  - int, float, bool, string, *tuple*
  - *not*: list, set, dictionary
- The set itself is **mutable** (e.g. we can add and remove elements)
- **Aside:** *frozenset* must contain immutable values and is itself immutable (cannot add and remove elements)