SETS

SETS

- in mathematics, a set is a collection of objects, potentially of many different types
  - in a set, no two elements are identical. That is, a set consists of unique elements
  - there is no order to the elements of a set
  - a set with no elements is the empty set

SETS

- A set can be created in one of two ways:

These two lines of code will create identical sets

```
1   my_set = set('abc')
2
3   my_set = {'a', 'b','c'}
```

Notice that sets use the curly braces just like dictionaries

SETS

- How do we create an empty set?

We create an empty set like this

```
my_set = set()
```

A common mistake is to think that creating an empty set is done like this. That is wrong. This will create an empty dictionary

```
my_set = {}
```

SETS

- A set can consist of a mixture of different types of elements

This set contains a string, an integer, a float and a boolean

```
1    my_set = { 'a', 1, 3.14159, True }
```

SETS

- duplicates are automatically removed from sets

```
1    my_set = set("aabbccdd")
2    print(my_set)   # prints {'a', 'c', 'b', 'd'}
```

SETS

- Here is an example from the course textbook

```
>>> null_set = set()          # set() creates the empty set
>>> null_set
set()
>>> a_set = {1,2,3,4}         # no colons means set
>>> a_set
{1, 2, 3, 4}
>>> b_set = {1,1,2,2,2}       # duplicates are ignored
>>> b_set

{1, 2}
>>> c_set = {'a', 1, 2.5, (5,6)} # different types is OK
>>> c_set
{(5, 6), 1, 2.5, 'a'}
>>> a_set = set("abcd")       # set constructed from iterable
>>> a_set
{'a', 'c', 'b', 'd'}          # order not maintained!
```

SETS

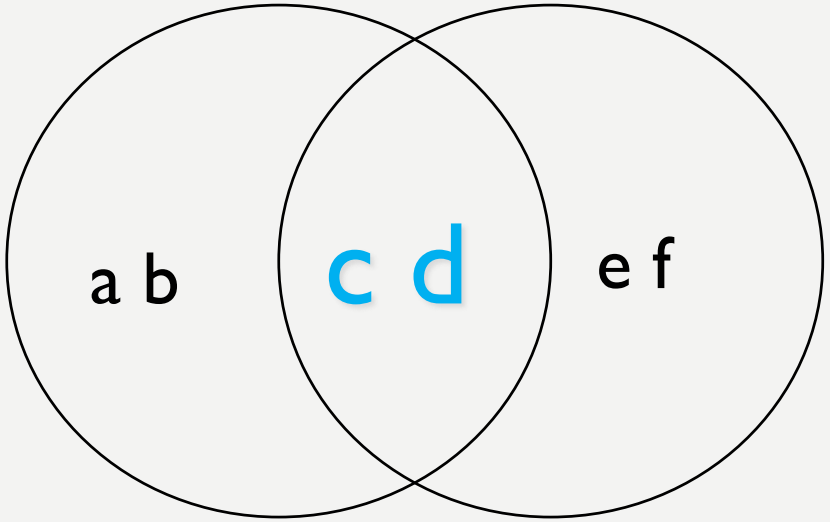
- common operators for sets
  - Sets respond to these functions like expected:
  - `len(my_set)`
    - the number of elements in a set
  - `element in my_set`
    - boolean indicating whether `element` is in my_set or not
  - `for element in my_set:`
    - iterate through the elements in `my_set`

```
1   my_set = set("aabbccdd")
2
3   length = len(my_set)
4
5   print(length) # prints 4
6
7   if 'a' in my_set:
8       print("a is in my_set")
9
10  for element in my_set:
11      print(element)
```

SETS

- Set operators
  - The set data structure provides some special operators that correspond to the operators you learned in middle school
  - These are various combinations of set contents
  - These operations have both a method name and a shortcut binary operator

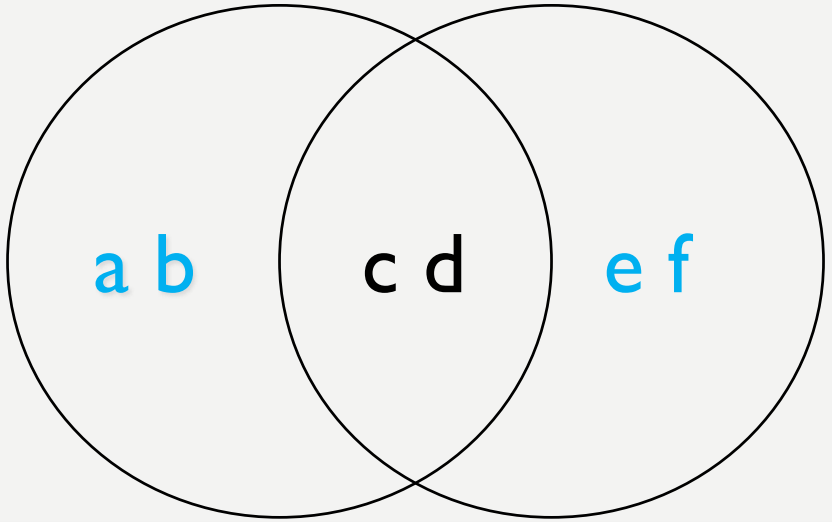
SETS
INTERSECTION



The & is shorthand for the .intersection method. So these two variables will contain the same resulting set.

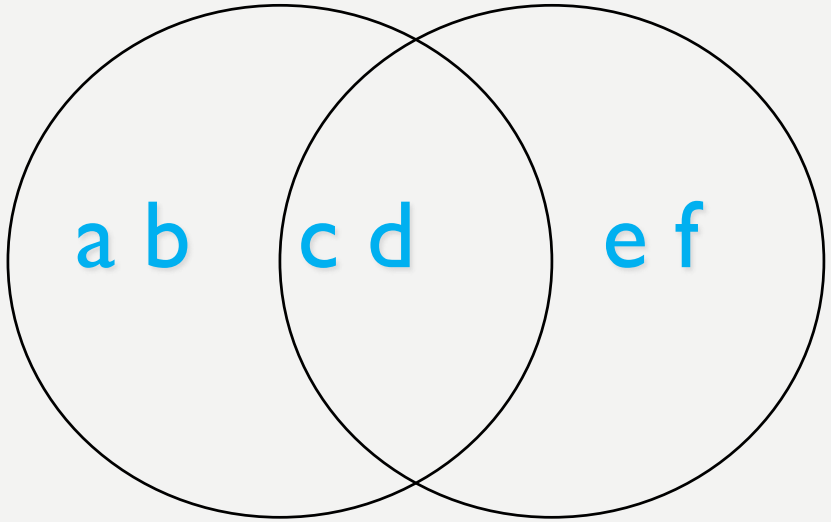
```
1   a_set = {'a', 'b', 'c', 'd'}
2   b_set = {'c', 'd', 'e', 'f'}
3
4   intersection1 = a_set & b_set # {'c', 'd'}
5   intersection2 = b_set.intersection(a_set) # {'c', 'd'}
```

SETS
DIFFERENCE



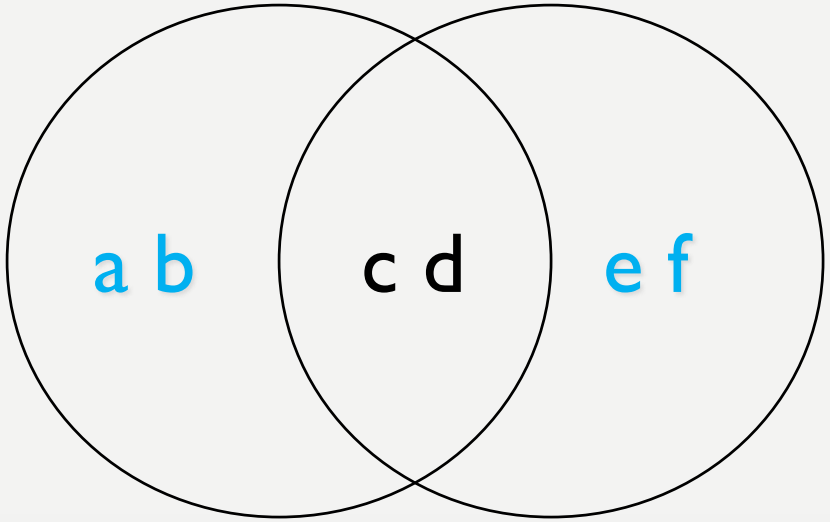

```
1   a_set = {'a', 'b', 'c', 'd'}
2   b_set = {'c', 'd', 'e', 'f'}
3
4   difference1 = a_set - b_set # {'a', 'b'}
5   #a_set.difference(b_set) is the same as in the line above
6
7   difference2 = b_set.difference(a_set) # {'e', 'f'}
8   # b_set - a_set #  {'e', 'f'} is the same as in the line above
```

SETS
UNION



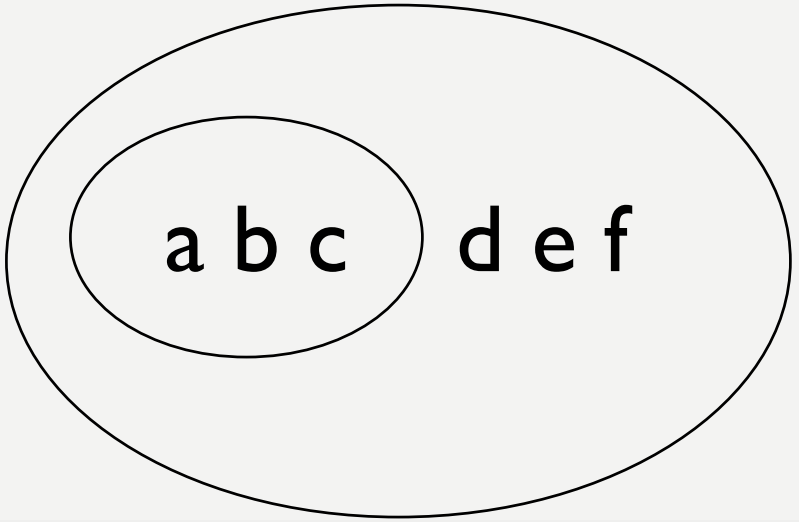

```
1   a_set = {'a', 'b', 'c', 'd'}
2   b_set = {'c', 'd', 'e', 'f'}
3
4   union01 = a_set | b_set # {'a', 'b', 'c', 'd', 'e', 'f'}
5   union02 = b_set.union(a_set) # {'a', 'b', 'c', 'd', 'e', 'f'}
```

SETS
SYMMETRIC_DIFFERENCE




```
1   a_set = {'a', 'b', 'c', 'd'}
2   b_set = {'c', 'd', 'e', 'f'}
3
4   result1 = a_set ^ b_set # {'a', 'b', 'e', 'f'}
5   result2 = b_set.symmetric_difference(a_set) # {'a', 'b', 'e', 'f'}
```

SETS
ISSUBSET AND ISSUPERSET




```
small_set= { 'a', 'b', 'c'}
big_set= { 'a', 'b', 'c', 'd', 'e', 'f' }

if small_set <= big_set:
    print("small set is a subset of bit set")
if big_set >= small_set:
    print("big set is a superset of small set")
```

SETS

- Other methods on sets
  - `my_set.add("g")` adds to the set, no effect if item is in set already
  - `my_set.clear()` emptys the set
  - `my_set.remove("g")` removes `"g"` from the set but throws an error if `"g"` isn't there
  - `my_set.discard("g")` removes `"g"` from the set. Nothing happens if `"g"` isn't in the set
  - `my_set.copy()` returns a shallow copy of `my_set`

SET COMPREHENSION

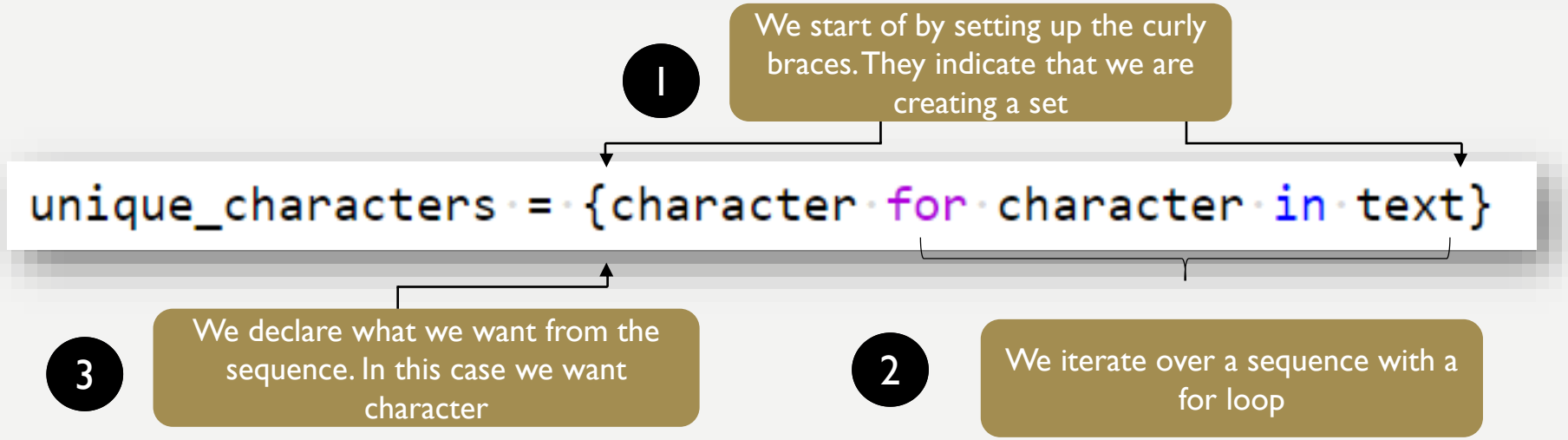
- We can create sets by using set comprehension
  - This works like list comprehension

```
1   text = "to be or not to be"
2
3   unique_characters = {character for character in text}
4
5   print(unique_characters) # prints {'r', ' ', 't', 'e', 'n', 'b', 'o'}
```

SET COMPREHENSION

- Here is an explanation

```
1  text = "to be or not to be"
2
3  unique_characters = {character for character in text}
4
5  print(unique_characters) # prints {'r', ' ', 't', 'e', 'n', 'b', 'o'}
```



SETS

- Common words in Gettysburg Address and Declaration of Independence
  - can reuse or only slightly modify much of the code for document frequency
  - the overall outline remains much the same
  - for clarity, we will ignore any word that has three characters or less (typically stop words)

SETS

- 4 functions
  - `add_word(word, word_set)`. Add word to the set (instead of dict). No return.
  - `process_line(line, word_set)`. Process line and identify words. Calls `add_word`. No return. (no change except for parameters)
  - `pretty_print(word_set)`. Nice printing of the various set operations. No return
  - `main()`. Function to start the program.

SETS

```
1 def add_word(word, word_set):
2     '''Add the word to the set. No word smaller than length 3.'''
3     if len(word) > 3:
4         word_set.add(word)
```

SETS

```
1  import string
2  def process_line(line, word_set):
3      '''Process the line to get lowercase words to be added to the set.'''
4      line = line.strip()
5      word_list = line.split()
6      for word in word_list:
7          # ignore the '--' that is in the file
8          if word != '--':
9              word = word.strip()
10             # get commas, periods and other punctuation out as well
11             word = word.strip(string.punctuation)
12             word = word.lower()
13             add_word(word, word_set)
```

SETS

- more complicated pretty print
  - the `pretty_print` function applies the various set operators to the two resulting sets
  - prints, in particular, the intersection in a nice format
  - should this have been broken up into two functions??

```
def pretty_print(ga_set, doi_set):
    # print some stats about the two sets
    print('Count of unique words of length 4 or greater')
    print('Gettysburg Addr: {}, Decl of Ind: {}\n'.format(len(ga_set),len(
doi_set)))
    print('{:15s} {:15s}'.format('Operation', 'Count'))
    print('-'*35)
    print('{:15s} {:15d}'.format('Union', len(ga_set.union(doi_set))))
    print('{:15s} {:15d}'.format('Intersection', len(ga_set.intersection(
doi_set))))
    print('{:15s} {:15d}'.format('Sym Diff', len(ga_set.symmetric_difference(
doi_set))))
    print('{:15s} {:15d}'.format('GA-DoI', len(ga_set.difference(doi_set))))
    print('{:15s} {:15d}'.format('DoI-GA', len(doi_set.difference(ga_set))))

    # list the intersection words, 5 to a line, alphabetical order
    intersection_set = ga_set.intersection(doi_set)
    word_list = list(intersection_set)
    word_list.sort()
    print('\n Common words to both')
    print('-'*20)
    count = 0
    for w in word_list:
        if count % 5 == 0:
            print()
        print('{:13s}'.format(w), end=' ')
        count += 1
```

```
--- inner local namespace ---
outer_var:34
inner_var:95
param_Inner:34

--- outer local namespace ---
outer_var:34
param_Outer:7
result:95
inner_function:<function inner_function at 0xe2ba30>

--- result ---
Result: 95
```