COMP10001

Week 6
With Lucy

First, some revision!

Do the following snippets do the same thing?
What are their advantages/disadvantages?

```
print("We need some saws")
print("We need some hammers")
print("We need some cogs")
print("We need some nails")
```

```
def get_str(part):
    return f"We need some {part}"

print(get_str("saws"))
print(get_str("hammers"))
print(get_str("cogs"))
print(get_str("nails"))
```

```
def get_str(part):
    return f"We need some {part}"

parts = ("saws", "hammers", "cogs", "nails")

for part in parts:
    print(get_str(part))
```

Consider the following loops. Are the two for loops equivalent? Why might you choose one over the other?

```
count = 0
items = ('eggs', 'spam', 'more eggs')
while count < len(items):
    print(f"we need to buy more {items[count]}")
    count += 1
```

```
items = ('eggs', 'spam', 'more eggs')
for count in range(len(items)):
    print(f"we need to buy more
{items[count]}")
```

```
items = ('eggs', 'spam', 'more eggs')
for item in items:
    print(f"we need to buy more
{item}")
```

Dictionaries and sets

When should we use a dictionary? How is it structured? How do we add and delete items?

When should we use a dictionary? How is it structured? How do we add and delete items?

- Dictionaries hold pairs of keys and values

When should we use a dictionary? How is it structured? How do we add and delete items?

- Dictionaries hold pairs of keys and values
- Useful for counting frequencies, storing information about unique items
- Access values with index notation: dictionary_name[key] gives value

When should we use a dictionary? How is it structured? How do we add and delete items?

- Dictionaries hold pairs of keys and values
- Useful for counting frequencies, storing information about unique items
- Access values with index notation: dictionary_name[key] gives value
- Add values by declaring a key value pair: dictionary_name[key] = value
- Remove values with .pop(): dictionary_name.pop(key)

What is the difference between using the .pop() method on a dictionary and using it on a list?

- List

- Dictionary

What is the difference between using the .pop() method on a dictionary and using it on a list?

- List
  - .pop()
  - .pop(index)
- Dictionary

What is the difference between using the .pop() method on a dictionary and using it on a list?

- List
  - .pop()
  - .pop(index)
- Dictionary
  - .pop(key)
  - .pop()

What is the difference between using the .pop() method on a dictionary and using it on a list?

- List
  - .pop() called without an index argument removes the last item in the list
  - .pop(index) removes the item in the list at that index
- Dictionary
  - .pop(key)
  - .pop()

What is the difference between using the .pop() method on a dictionary and using it on a list?

- List
  - .pop() called without an index argument removes the last item in the list
  - .pop(index) removes the item in the list at that index
- Dictionary
  - .pop(key) deletes the key:value pair associated with that key in a dictionary
  - .pop() called without a key on a dictionary raises an error

When should we use a set? How do sets differ from lists and dictionaries?

When should we use a set? How do sets differ from lists and dictionaries?

- Sets store collections of unique objects

When should we use a set? How do sets differ from lists and dictionaries?

- Sets store collections of unique objects
- E.g. use to remove duplicates from a list

When should we use a set? How do sets differ from lists and dictionaries?

- Sets store collections of unique objects
- E.g. use to remove duplicates from a list
- Also use to get access to set operations!

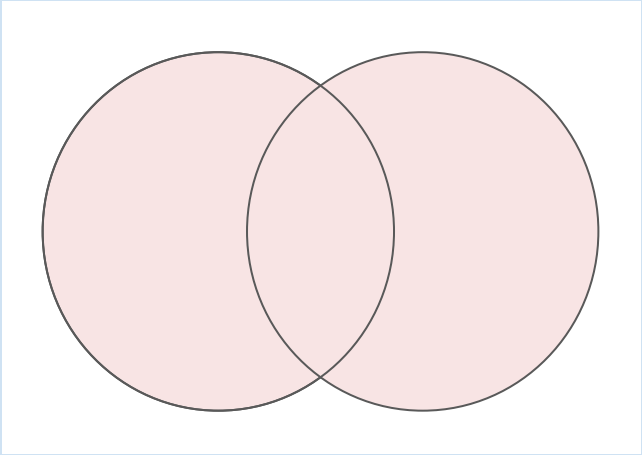
What operations can you perform on sets? How do you add and remove items from sets?

What operations can you perform on sets? How do you add and remove items from sets?

- Union:
- Intersection:
- Difference:

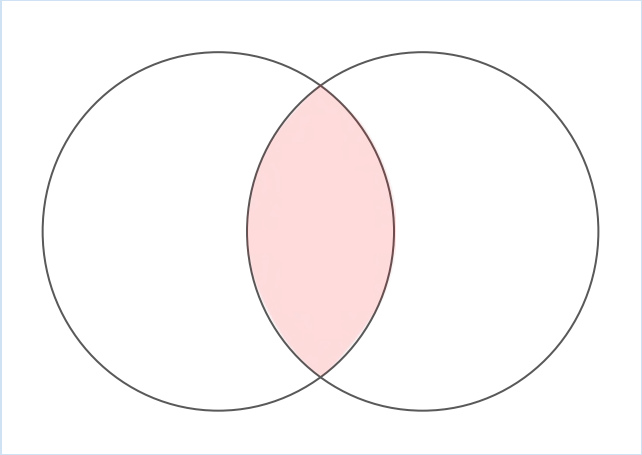
What operations can you perform on sets? How do you add and remove items from sets?

- Union: `s1 | s2` or `s1.union(s2)`
- Intersection:
- Difference:



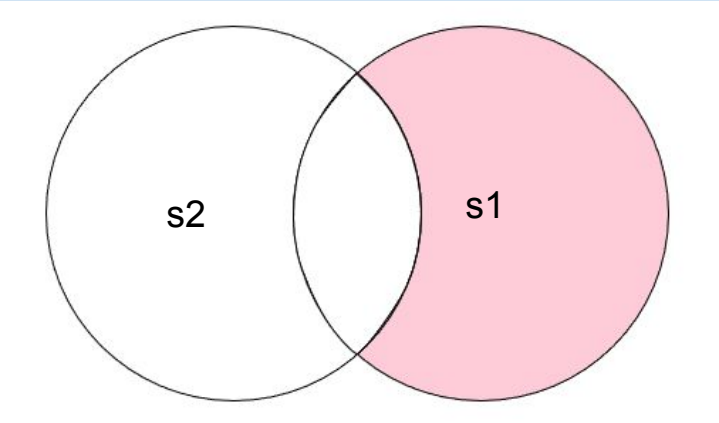
What operations can you perform on sets? How do you add and remove items from sets?

- Union: `s1 | s2` or `s1.union(s2)`
- Intersection: `s1 & s2` or `s1.intersection(s2)`
- Difference:



What operations can you perform on sets? How do you add and remove items from sets?

- Union: `s1 | s2` or `s1.union(s2)`
- Intersection: `s1 & s2` or `s1.intersection(s2)`
- Difference: `s1 - s2` or `s1.difference(s2)`



What operations can you perform on sets? How do you add and remove items from sets?

- Union: `s1 | s2` or `s1.union(s2)`
- Intersection: `s1 & s2` or `s1.intersection(s2)`
- Difference: `s1 - s2` or `s1.difference(s2)`

- Adding an item: `my_set.add(item)`
- Removing an item: `my_set.remove(item)`

Exercises!

What is `None`? How is it used?

What is `None`? How is it used?

- None is a special value:
  - The thing that's returned when a function has no return statement
  - Value we get when we assign to many mutating methods, e.g. .append()

What is `None`? How is it used?

- None is a special value:
  - The thing that's returned when a function has no return statement
  - Value we get when we assign to many mutating methods, e.g. .append()
- Represents the *absence of a result*

What is `None`? How is it used?

- None is a special value:
  - The thing that's returned when a function has no return statement
  - Value we get when we assign to many mutating methods, e.g. .append()
- Represents the *absence of a result*
- Its own type (not really Boolean!) so no value of any other type has equality with it!

What is the difference between sorted() and .sort() when applied to a list?

What is the difference between sorted() and .sort() when applied to a list?

- Both `sorted()` and `.sort()` are able to sort lists

What is the difference between sorted() and .sort() when applied to a list?

- Both `sorted()` and `.sort()` are able to sort lists
- `sorted(my_list)` returns a new list, containing the items in `my_list`, sorted
- `my_list.sort()` does not return anything (its value is `None`)

What is the difference between sorted() and .sort() when applied to a list?

- Both `sorted()` and `.sort()` are able to sort lists
- `sorted(my_list)` returns a new list, containing the items in `my_list`, sorted
- `my_list.sort()` does not return anything (its value is `None`)
- `sorted(my_list)` does not change the value of the variable `my_list`
- `my_list.sort()` changes the value of the variable `my_list`

What is the difference between sorted() and .sort() when applied to a list?

- Both `sorted()` and `.sort()` are able to sort lists
- `sorted(my_list)` returns a new list, containing the items in `my_list`, sorted
- `my_list.sort()` does not return anything (its value is `None`)
- `sorted(my_list)` does not change the value of the variable `my_list`
- `my_list.sort()` changes the value of the variable `my_list`

- `.sort()` changes the value of `my_list` ***in-place***

What is the difference between sorted() and .sort() when applied to a list?

- For example:

```
lst = [2,1,3]
new_lst = lst.sort()
print(lst, new_lst)
```

```
lst = [2,1,3]
new_lst = sorted(lst)
print(lst, new_lst)
```

What is the difference between sorted() and .sort() when applied to a list?

- For example:

```
lst = [2,1,3]
new_lst = lst.sort()
print(lst, new_lst)
```

```
[1,2,3] None
```

```
lst = [2,1,3]
new_lst = sorted(lst)
print(lst, new_lst)
```

```
[2,1,3] [1,2,3]
```

Exercises!

Write a function with takes a string as inputs and print the frequency of each character in the string, using a dictionary

E.g. `freq_counts("booboo")`
should print:

```
b 2

o 4
```

Write a function with takes a string as inputs and print the frequency of each character in the string, using a dictionary

E.g. `freq_counts("booboo")`
should print:

```
b 2

o 4
```

```
def freq_count(words):
    freqs = {}
    for letter in words:
        if letter in freqs:
            freqs[letter] += 1
        else:
            freqs[letter] = 1
    for key in freqs:
        print(key, freqs[key])
```

Write a function which takes two lists as input and returns a list containing the numbers which they both have in common

E.g., `in_common([1,2,4], [3,4,5])` should return `[4]`

Write a function which takes two lists as input and returns a list containing the numbers which they both have in common

E.g., `in_common([1,2,4], [3,4,5])` should return `[4]`

```
def in_common(list_1, list_2):
    set_1 = set(list_1)
    set_2 = set(list_2)
    common = set_1 & set_2
    return list(common)
```