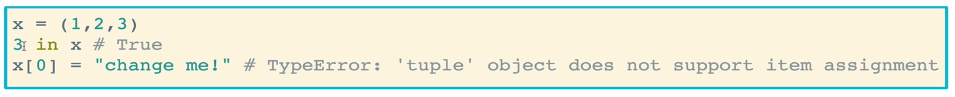
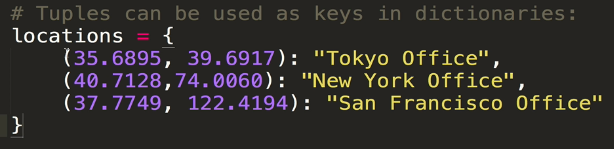
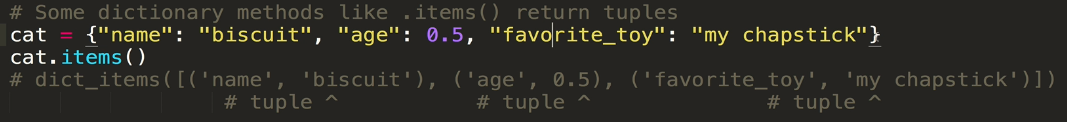
* A **tuple** is an ordered collection or grouping of items
  + The items can be numbers, strings, Booleans, or a mix of any of these
  + It’s different from a list in the following ways:
    - Syntax – uses parentheses instead of brackets
    - Immutability – it cannot be changed, ever! You cannot add things or remove things



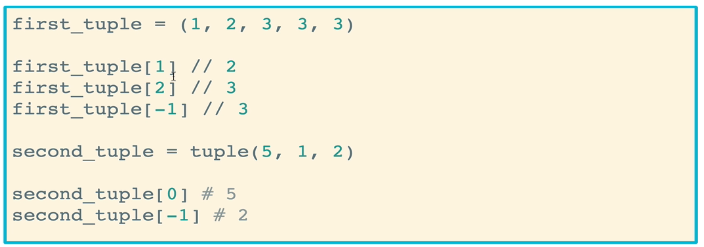
* + Tuples are useful because
    - They are lighter weight and faster than lists, less computationally expensive
    - Makes your code safer from bugs or unforeseen issues popping up, since they can’t be changed
    - Tuples can be used as valid keys in a dictionary
      * A single tuple of many values can be considered one key
      * Lists *cannot* be used as keys in a dictionary



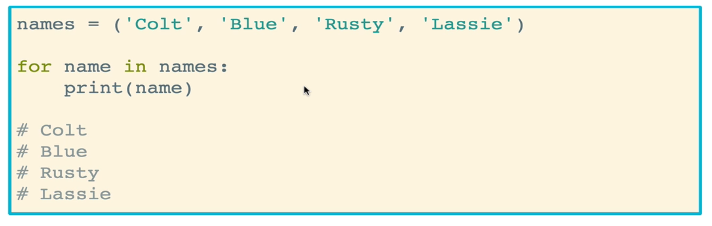
* + - Some methods return tuples to you
      * Such as the .items() method for dictionaries



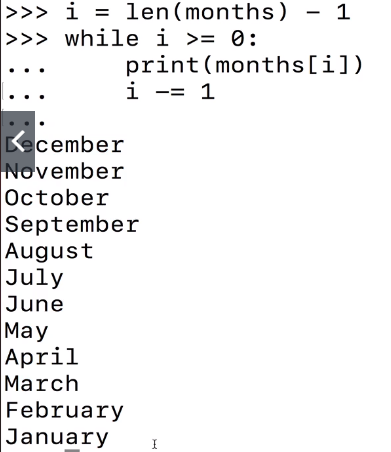
* Create tuples using parentheses or the **tuple**() function
  + Use parentheses to create tuples from scratch
  + Use the tuple function to convert other types of objects to tuples



* Tuple methods
  + We can iterate over tuples using the same syntax that we use for iterating over lists
    - For loop:



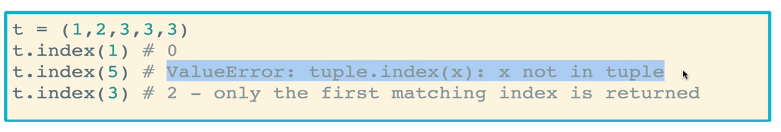
* + - While loop:



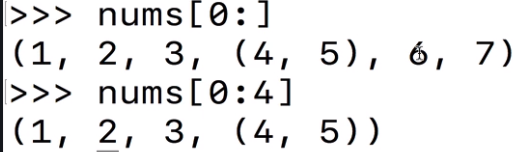
* + .**count**() returns the number of times that a value appears in a tuple



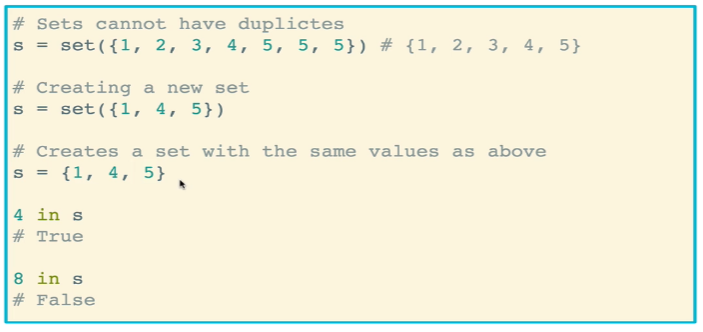
* + .**index**(*value*) returns the index at which a value is found in a tuple
    - If a value is not in there, a ValueError is thrown
    - If the value appears multiple times in the tuple, only the first matching index is returned



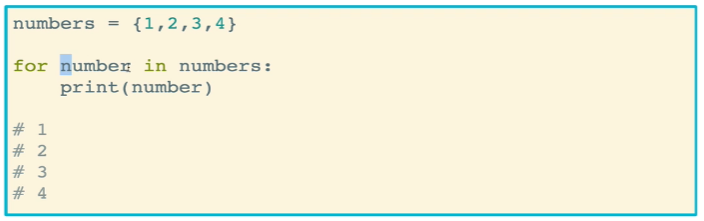
* Tuples can be nested within other tuples
  + To access values within deeper tuples, you can use index notation just like lists
* You can also slice tuples



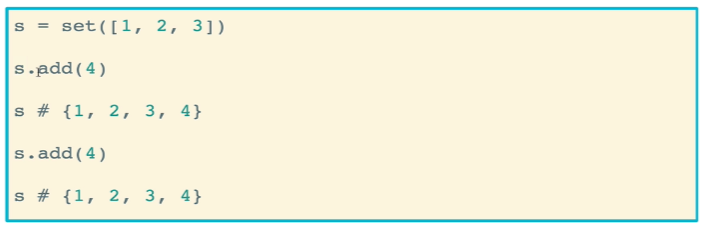
* A **set** is like a formal mathematical set
  + Sets do not have duplicate values
  + Sets are unordered
  + Think of it as a group of whatever (numbers, strings, Booleans, etc.) with no order and no duplicates
  + You cannot access them by indices
  + Useful to keep track of elements but don’t care about ordering or keys, but DO want to make sure that there are no duplicates
  + Pro tip, you can convert a list to a set in order to remove duplicates! This is a good use case
    - set(*list*)
* Sets are created using curly braces *without* the colon syntax
  + Note that if you try to create a set with duplicates, the set that gets created will only have one of the attempted duplicate item
  + You can use logical testing to check whether a particular item is in the set



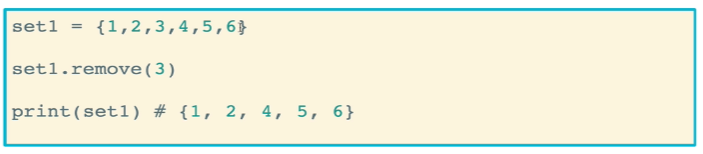
* You can access all values of a set using a trusty for loop



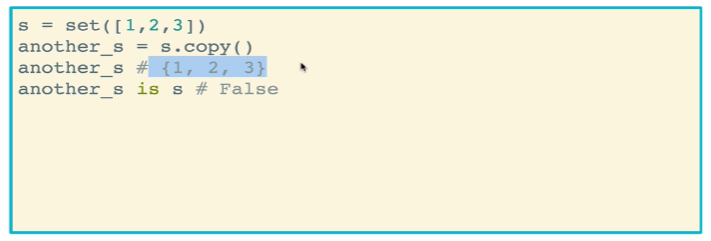
* Set methods
  + The **add** method allows you to add data to a set as you go
    - If the set already contains the item you are adding, no error will be thrown. Rather, it simply will not be added



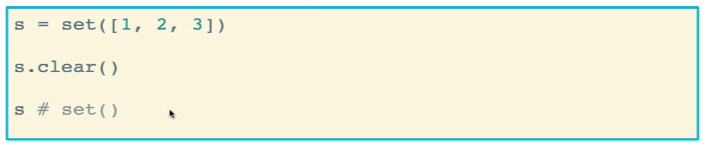
* + The **remove** method removes a value from as set
    - If you attempt to remove an item that is not in a set, an error will be thrown
    - A **.discard(*item*)** method can alternatively be used to avoid throwing the error



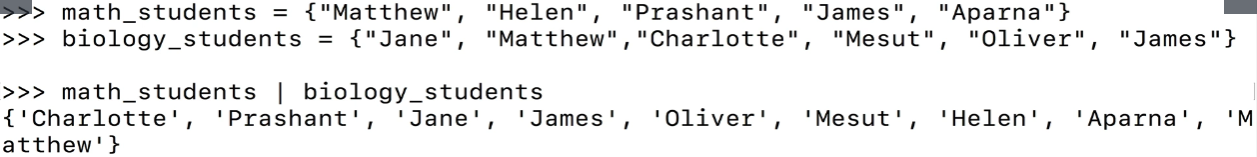
* + The **copy** method makes a copy of the set
    - Similar to lists and dictionary, it will make a duplicate of the set, but that duplicate will be a unique item in memory



* + The **clear** method removes the entire contents of a set



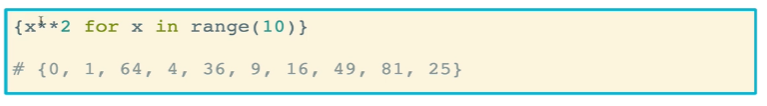
* Set mathematical functions
  + The **union** method is accessed using the pipe syntax (set1 | set2). It simply creates a set that consists of elements from both sets. Remember that the created set will only have one copy of each element



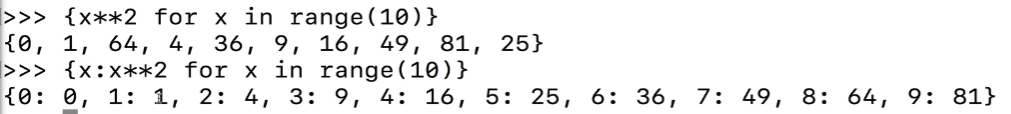
* + The intersection method returns a set containing items that belong to both sets being compared
    - It is accessed using the ampersand (set1 & set2)



* **Set** **comprehension** is similar to list and dictionary comprehension in that you can quickly create new sets using list comprehension
  + Unlike dictionaries, you don’t identify a *key,value* pair because there are none. Instead, you simply identify the item in the set
  + Remember that the set that is created has no order



* + You can, however, use key:item notation to generate a dictionary from a set!



* + You can also perform set comprehension on things that are not sets, like lists and strings. Keep in mind that any duplicates will be removed and the order (if present in the original) will not be maintained

