**Logo

Description automatically generated**

**Concatenating lists together**

*# 3. TODO: Define our\_list by concatenating my\_list and your\_list*

our\_list = my\_list + your\_list

**Appending lists**

*# 4. TODO: Append their\_flower to the end of our\_list*

our\_list.append(their\_flower)

**Replacing lists items (2 methods):**

(<https://blog.finxter.com/how-to-use-rangelen-in-python/>) this is a good article for later. But, the quote the blog here, “Generally, range(len()) allows you to iterate across a given iterable/sequence to access each item of the sequence with the help of its index.”

1. Text

   Description automatically generated**Range(len) method:** this one looks at the items of our list like [1,2,3,4,5] or [mango, strawberry, grape, watermelon, orange, apple]. It starts on the left and starts iterating or walking to the right for length of the list. In the number example len = 5, for the fruit it is len =6. So, for the fruit example it will start with mango and keep walking to apple, unless you tell it to stop. Below, I’ve told it to stop iterating if it finds my\_flower2. I had to use == because it was basically a Boolean argument (if this condition is true, then do this). In this case, the condition was met with my\_flower2 at whatever iteration it was on. We don’t care what [i] right now, all we care is that at that location, we told the program to stop iterating. Then, we told it to basically to a variable substitution. So, our\_list[i] is where our program stopped and we are going to substitute their\_flower for that location.

**\*\*This method shouldn’t be used in zybooks right now, because it can create an assignment split at no fault of the operator. Kinda like if I had two couples on a double date who wanted to switch partners, but instead, it created two identical girls in the swapped group. I would use the Brandon method instead.**

1.  **Brandon’s method:** So, I got this from Brandon and I really liked it. Rather than iterating through a list. This method just says, on our list our\_list at our indexed location my\_flower2, do a variable swap with their\_flower, just like above. You might be asking, why don’t we just put our\_list[my\_flower2] = their\_flower. We when we use [i 1,2,3,4 or whatever] it expects an indexed location. It’s like say to our\_list[2], GO to the 3rd spot from the left. So, whatever is inside the [brackets] needs to be some kind of indexed location. That’s where .index come in. The index() method returns the index of the given element in the list. For instance:

list2 **=** ['cat', 'bat', 'mat', 'cat', 'pet']

**print**(list2.index('bat'))

**Output: 1 (“bat” is located at position 1)**

So, in Brandon’s example, we use the .index to find the position of my\_flower2 and swap it with their flower.

**Removing the first occurrence of an element of a list. Use remove()**

our\_list.remove(their\_flower)

**Removing an indexed item from the list.**

our\_list.pop(1)

**Sets**

The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set. This is based on a data structure known as a hash table. Since sets are unordered, we cannot access items using indexes as we do in lists.

1. The set doesn’t maintain elements in any particular order.
2. Only instances of immutable types can be added to a Python set.

Creating sets

var = {“marbles”, “cats”, “tails”, “gimmicks”}

Turning a list into a set

myset = set([“a”,”b”,”c”])

myset.add(“d”)

print(myset) {“d”,“a”,”b”,”c}

frozen sets are immutable

frozen\_set = frozenset([“e”, “f”, “g”])

if you tried to run frozen\_set.add(“a”) it would return an error

adding to sets

people **=** {"Jay", "Idrish", "Archi"}

# Creating a Set

people **=** {"Jay", "Idrish", "Archi"}

people.add("Daxit")

# Adding elements to the

# set using iterator

**for** i **in** range(1, 6):

    people.add(i)

print("\nSet after adding element:", end **=** " ")

print(people)

Union

Two sets can be merged using Union

people **=** {"Jay", "Idrish", "Archil"}

vampires **=** {"Karan", "Arjun"}

dracula **=** {"Deepanshu", "Raju"}

# Union using union()

# function

population **=** people.union(vampires

Union using union() function

{'Karan', 'Idrish', 'Jay', 'Arjun', 'Archil'}

Intersection

|  |
| --- |
| # Python program to  # demonstrate intersection  # of two sets    set1 **=** set()  set2 **=** set()    **for** i **in** range(5):      set1.add(i)    **for** i **in** range(3,9):      set2.add(i)    # Intersection using  # intersection() function  set3 **=** set1.intersection(set2)    print("Intersection using intersection() function")  **print**(set3)    # Intersection using  # "&" operator  set3 **=** set1 & set2    print("\nIntersection using '&' operator")  **print**(set3) |

Intersection using intersection() function

{3, 4}

Difference

|  |
| --- |
| Python program to  # demonstrate difference  # of two sets    set1 **=** set()  set2 **=** set()    **for** i **in** range(5):      set1.add(i)    **for** i **in** range(3,9):      set2.add(i)    # Difference of two sets  # using difference() function  set3 **=** set1.difference(set2)    print(" Difference of two sets using difference() function")  **print**(set3) |

Remove items from set

|  |
| --- |
| # Python program to demonstrate  # Deletion of elements in a Set    # Creating a Set  set1 **=** set([1, 2, 3, 4, 5, 6,              7, 8, 9, 10, 11, 12])  **print**("Initial Set: ")  print(set1)    # Removing elements from Set  # using Remove() method  set1.remove(5)  set1.remove(6)  **print**("\nSet after Removal of two elements: ")  **print**(set1)    # Removing elements from Set  # using Discard() method  set1.discard(8)  set1.discard(9)  print("\nSet after Discarding two elements: ")  **print**(set1)    # Removing elements from Set  # using iterator method  **for** i **in** range(1, 5):      set1.remove(i)  print("\nSet after Removing a range of elements: ")  print(set1) |

Initial Set:

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

Set after Removal of two elements:

{1, 2, 3, 4, 7, 8, 9, 10, 11, 12}

Set after Discarding two elements:

{1, 2, 3, 4, 7, 10, 11, 12}

Set after Removing a range of elements:

{7, 10, 11, 12}

|  |
| --- |
| # List  list\_of\_items **=** ['g', 'e', 'e', 'k', 's']  print(sorted(list\_of\_items))  ['e', 'e', 'g', 'k', 's']    # Tuple  tuple\_of\_items **=** ('g', 'e', 'e', 'k', 's')  print(sorted(tuple\_of\_items))  ['e', 'e', 'g', 'k', 's']    # String-sorted based on ASCII  # translations  string **=** "geeks"  **print**(sorted(string))  ['e', 'e', 'g', 'k', 's']    # Dictionary  dictionary **=** {'g': 1, 'e': 2, 'k': 3, 's': 4}  print(sorted(dictionary))  ['e', 'g', 'k', 's']    # Set  set\_of\_values **=** {'g', 'e', 'e', 'k', 's'}  **print**(sorted(set\_of\_values))  ['e', 'g', 'k', 's']    # Frozen Set  frozen\_set **=** frozenset(('g', 'e', 'e', 'k', 's'))  print(sorted(frozen\_set))  ['e', 'g', 'k', 's'] |