```
"""
                    PYTHON LISTS

A list is a data structure that stores items in an ordered manner. The items can
be of varying types, duplicates are allowed and the items are indexed.
Lists are MUTABLE meaning the items can be changed and deleted

Lists can be declared with square brackets and can also be created using the
list() constructor passing in another iterable object
List items are seperated by commas ','
"""
# The 'copy' module is used at the end of the notes for the 'Copying Lists Section'
import copy

# declare a list
num_list = [1, 2, 3]
animal_list = ["dog", "cat", "sheep"]

# lists can be declared as empty
my_list = []

# using the '*' operator, we can populate a list quickly:
quick_list = ["2"] * 4
print(quick_list)                   # ['2', '2', '2', '2']

# NOTE: Be careful where the '*' goes:
second_quick_list = ["2" * 4]
print(second_quick_list)            # ['2222']

third_quick_list = [1*4]            # [4]
print(third_quick_list)             # [1, 1, 1, 1]

fourth_quick_list = [1] * 4
print(fourth_quick_list)

# lists can hold items of different types (including other lists)
car_description = ['Audi', 'R8', 5.2, ['blue', 'red', 'green', 'black', 'silver']]

# consider creating a list from a tuple
topic_tuple = ("trigonometry", "geometry", "algebra")
topic_list = list(topic_tuple)
print(type(topic_list))                 # <class 'list'>
print(topic_list)                       # ['trigonometry', 'geometry', 'algebra']

"""
============================================================
Add items to a list
"""

# 1. append(item) - adds an item to the end of the list
even_numbers = [2, 4, 6]
even_numbers.append(8)
print(even_numbers)             # [2, 4, 6, 8]

    # Be careful when appending another list, this will add the list into one
    # index position of the original
second_even_list = [10, 12, 14]
even_numbers.append(second_even_list)
print(even_numbers)             # [2, 4, 6, 8, [10, 12, 14]]

# 2. insert(index_position, new_item) -> insert an item into a specified index position of the list
odd_list = [1, 3, 7, 9, 11]
# insert the number 5, into index position 2
odd_list.insert(2, 5)
print(odd_list)                 # [1, 3, 5, 7, 9, 11]

"""
============================================================
Extending and combining lists
"""

# lists can be combined with the plus '+' operator
first_list = [1, 2, 3]
second_list = [4, 5, 6]
```

```python
third_list = first_list + second_list
print(third_list)               # [1, 2, 3, 4, 5, 6]

    # NOTE: Attempting to combine a list with an non-list will
    # result in a TypeError

    # Combine a list with an Integer = BAD
    # powers_of_2 = [2, 4, 8, 16]
    # extended_2_powers = powers_of_2 + 32        -> Results in a TypeError

    # Combine a list with a Tuple =  BAD
    # powers_of_2 = [2, 4, 8, 16]
    # extended_2_powers = powers_of_2 + (32, 64)  -> Results in a TypeError

    # However, we can make use of the list() constructor before attempting to combine the tuple
powers_of_2 = [2, 4, 8, 16]
extended_2_powers = powers_of_2 + list((32, 64))
print(extended_2_powers)                            # [2, 4, 8, 16, 32, 64]


# lists can be extended using the extend() function. Extending a list means that the items of a
# new list are added in as individual items into the other list. This is not the same as append()

bird_list = ['pigeon', 'sparrow', 'eagle']
additional_birds = ['robin', 'raven']
bird_list.extend(additional_birds)
print(bird_list)                                # ['pigeon', 'sparrow', 'eagle', 'robin', 'raven']


    # BE CAREFUL, consider adding a nested list (list of lists)
    # the nested lists will remain nested
tree_list = ['Jacaranda', 'Oak', 'Redwood']
categorized_trees = [['apple', 'orange', 'lemon'], ['Cyathea', 'Boston Fern', 'Autumn Fern']]
tree_list.extend(categorized_trees)
print(tree_list)
# ['Jacaranda', 'Oak', 'Redwood', ['apple', 'orange', 'lemon'], ['Cyathea', 'Boston Fern', 'Autumn Fern']]


# The extend function can be used to add other iterable objects to a list, Consider the Dictionary below
    # NOTE: Only the keys from the dictionary are added to the list
laptop_list = ['Asus', 'Mac', 'Dell']
laptop_dict = {'Lenovo':'2019', 'HP':2022}
laptop_list.extend(laptop_dict)
print(laptop_list)                          # ['Asus', 'Mac', 'Dell', 'Lenovo', 'HP']

# NOTE: extend is looking for an iterable object (like a list), but if we supply a string then
# it will add each character to the list:
laptop_list.extend("Alienware")
print(laptop_list)
# ['Asus', 'Mac', 'Dell', 'Lenovo', 'HP', 'A', 'l', 'i', 'e', 'n', 'w', 'a', 'r', 'e']

# Attempting to use extend() with a non-iterable argument (such as an int) results in a TypeError
# laptop_list.extend(4)         # TypeError

"""
============================================================
Indexing a list and modifying an item
"""
# Lists are ordered and we can access List Items by using square brackets '[]' and specifying the
# index position of the item we are looking for.
# NOTE: Lists are indexed from 0 to the length of the list - 1
# NOTE: Attempting to access an index position not in the list will result in an IndexError

season_list = ['Summer', 'Autumn', 'Winter', 'Spring']
# access the first element
first_season = season_list[0]
print(first_season)                             # Summer
# acess the last element
print(season_list[len(season_list) - 1])        # Spring

# the list indexes are from 0 - 3, if we try access index 4 we will get an IndexError
# print(season_list[4])                         # IndexError

# REMEMBER that lists are Mutable (can be changed)
```

```python
# let's change the last element of the list
season_list[3] = 'Season of Flowers and Sun'
print(season_list)    # ['Summer', 'Autumn', 'Winter', 'Season of Flowers and Sun']

"""
============================================================
Negative Indexing
"""
# Consider the above list and say I write code as season = season_list[-1]
# In Python, this does not mean index position -1, it means return the last index position
# and is known as Negative Indexing

print(season_list[-1])                # Season of Flowers and Sun
print(season_list[-2])                # Winter
print(season_list[-3])                # Autumn
print(season_list[-4])                # Summer

# NOTE: There is a limit, you cannot negatively index beyond the first element such as
# print(season_list[-5])                # IndexError

"""
============================================================
List Slicing
"""
# Consider wanting to create a new list from specific items within a list, we
# can use List Slicing

# List Slicing uses Square Brackets with three values seperated by colons
    # list_name[start_index : end_index : step]
    # NOTE: the start index is included, the end index is excluded

# Consider a list of tens
tens_list = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# 1) If we only wanted values from the second item to the fifth item (including), we can use list slicing
    # REMEMBER:
    # The second item has index position 1
    # The fifth item has index position 4, but we need to specify the end index position 5 in the list slice
    # as the end_index is not included. In other words we are saying go up to (but not including)
    # index position 5
    # we set the step as 1 to increment by 1 as we move along the list
shortened_tens = tens_list[1:5:1]           # start = 1, end = 5, step = 1
print(shortened_tens)                       # [20, 30, 40, 50]


# 2) To specify from the beginning, we simply leave the start index blank
print(tens_list[:5:1])               # [10, 20, 30, 40, 50]

    # we could also state the start index as 0
print(tens_list[0:5:1])               # [10, 20, 30, 40, 50]


# 3) To specify 'up to the end' we leave the end index blank
print(tens_list[2::1])                # [30, 40, 50, 60, 70, 80, 90, 100]

    # you could also state the end_index = length of the list
    # does not cause an IndexError as the end_index is not included

print(tens_list[2:10:1])              # [30, 40, 50, 60, 70, 80, 90, 100]

    # if we set an end index beyond the length of the list, this does not cause an IndexError,
    # but instead returns up to the end of the list
print(tens_list[:15:1])               # [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# 4) Now, we can change the step value to count in two's
    # NOTE: This will begin at the start_index and then move in two's from there
print(tens_list[::2])                 # [10, 30, 50, 70, 90]

    # Not stating the step, will set it to the default value of 1
print(tens_list[1:5])                 # [20, 30, 40, 50]

    # Leaving the step as blank will do the same thing
print(tens_list[1:5:])                # [20, 30, 40, 50]

```

```
"""
============================================================
List Slicing - Negative Step
This can be confusing, Take your time
"""

# 1) By setting the step to -1, we return a list in reverse order
reversed_list = tens_list[::-1]
print(reversed_list)                # [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]


# 2) By setting the step to -2, we return every second item from the list in reverse order
print(tens_list[::-2])              # [100, 80, 60, 40, 20]

    # Here we need to think deeper about what is happening:
    # 1) first look at the start index and end index
    # 2) Consider the direction, which is reversed (indicated by the minus sign on the step)
    # 3) Now Consider the step and go in reverse from end to start

    # 1) Start index is the beginning and the end index is the end (in this example)
    # 2) Go in reverse, so start at the end index and go up to the start index
    # 3) From the first value (which would be 100, count in twos):
    #       100, 80, 60, 40, 20

# 3) Now we try something like this:
print(tens_list[5:1:-1])            # [60, 50, 40, 30]

    # Breaking this down:
    # start at index 5 (the number 60) and end at index 1 (the number 20)
    # BUT remember that the end index is not included (so we will actually stop at 30 or index 2)
    # The sign is negative so we go in reverse with a step of 1
    # so start at 60 (index 5), then 50 (index 4), then 40 (index 3), then 30 (index 2)

# NOTE: we must make sure that when going in reverse, we set the start index > end index

"""
============================================================
Removing items from a list
"""

# pop(index) can remove an item from a list at the specified index. It also returns the removed item

noun_list = ["car", "book", "tree", "cloud"]
removed_item = noun_list.pop(1)
print(removed_item)                         # book
print(noun_list)                            # ['car', 'tree', 'cloud']

# if the argument is left blank, its default is -1 (the last item)
noun_list.pop()
print(noun_list)                            # ['car', 'tree']


# If the index given exceeds the length of the list, it will return an IndexError
# verb_list = ["run", "climb", "jump", "read", "talk"]
# verb_list.pop(6)                          # IndexError

    # If the index given is negative, it will pop from the end of the list
    # Here the item 'read' will be removed
verb_list = ["run", "climb", "jump", "read", "talk"]
verb_list.pop(-2)
print(verb_list)                        # ['run', 'climb', 'jump', 'talk']

    # With a negative index, we cannot exceed the first element
# adjective_list = ["happy", "excited", "sad", "angry"]
# adjective_list.pop(-5)                # IndexError


# The remove(item) method can be used to remove a specific item from a list. It does not
# return the removed item

article_list = ['The', 'A', 'An']
removed_article = article_list.remove('An')
print(removed_article)                          # None
print(article_list)                             # ['The', 'A']
```

```python

    # A ValueError is returned if the item is not in the list
# article_list.remove("Potato")                # ValueError

"""
============================================================
Checking for items in a list
"""
# 1) We can use the 'in' keyword to check if an item is in a list or not
# This will return a boolean

bag_content = ['pen', 'paper', 'calculator', 'exam pad']
pen_contained = 'pen' in bag_content
print(pen_contained)                    # True

# we can use this in if-statements
if 'calculator' in bag_content:
    print("Ready for Math Exam")
else:
    print("Borrow a Calculator")
# Will print -> Ready for Math Exam

if 'protractor' in bag_content:
    print("Ready for Geometry")
else:
    print("Need to get a Protractor")
# Will print -> Need to get a Protractor


# 2) we can use the index(item) method. If the item is in the list,
# it will return its index position, if not we will get a ValueError

# The code below does make use of a try-except statement to handle a ValueError
# and an f-string
study_hours = [3, 4, 4, 5, 6, 6]

try:
    first_long_study = study_hours.index(6)
    print(f"The student studied for 6 hours on day {first_long_study + 1}")
except ValueError:
    print("The student did not study for 6 hours this week")
# Will print -> The student studied for 6 hours on day 5

try:
    first_long_study = study_hours.index(8)
    print(f"The student studied for 8 hours on day {first_long_study + 1}")
except ValueError:
    print("The student did not study for 8 hours this week")
# Will print -> The student did not study for 8 hours this week

"""
============================================================
Lists and Loops, List Comprehension
"""
# Considering lists are iterable, we can use loops to move through them

# 1) Find the max value in an unsorted list (without using the max() method)

unsorted_nums = [10, -2, 5, 7, 1, 8, 15, -3]
# one-line if statement
max_number = unsorted_nums[0] if len(unsorted_nums) > 0 else None

if max_number is not None:
    for i in range(1, len(unsorted_nums)):
        if unsorted_nums[i] > max_number:
            max_number = unsorted_nums[i]

print(f"The max number was {max_number}")  # The max number was 15


# 2) Create a list containing the even numbers from another list
numbers_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
evens_list = []

```

```python
for number in numbers_list:
    if number % 2 == 0:
        evens_list.append(number)

print(evens_list)   # [2, 4, 6, 8, 10]


# 3) Repeat the above activity using List Comprehension
# List Comprehension is a shorted form of a for.. in loop that allows us to
# greatly condense the code needed to perform a task

# using the clear() method to remove all items from the evens_list
evens_list.clear()

# using list comprehension
evens_list = [i for i in numbers_list if i%2==0]
print(evens_list)   # [2, 4, 6, 8, 10]

"""
============================================================
List Methods

There are multiple methods than can be used on lists
We have already seen a few such as:
clear(), index(), pop(), insert(), append(), extend(), remove()
and here are some more
"""

# ------------------------------------------------------------------------------------------
# 1) max() -> find the largest value in a list

    # for numerical types, this works as expected, returning the largest number
float_list = [1.12, 2.54, -1.156, 3.75, 1.26]
print(f"Max of Float List is: {max(float_list)}")

    # for strings (default) the letters of the string are compared character by character
    # each character holds an integer value such as 'a' = 97
    # Consider 'dog' vs 'cat' -> 'd' = 100 and 'c' = 99 so d > c

word_list_1 = ["dog", "cat"]
print(max(word_list_1))             # dog

    # what about 'pack' vs 'pace'. The first three letters are the same so only the last letters
    # determine the max. With 'k' coming after 'e', the word 'pack' is greater
word_list_2 = ['pace', 'pack']
print(max(word_list_2))             # pack

    # now consider words with different lengths such as 'apple' vs 'ancient'
    # the first letters are the same, but then 'p' comes after 'n' (p > n) so the max is 'apple'
word_list_3 = ['apple', 'ancient']
print(max(word_list_3))             # apple

# BUT WHAT IF WE WANTED TO COMPARE THE STRINGS BASED ON THEIR LENGTH?
# The max() method has a parameter called 'key' in which we can specify how we want the values
# to be compared

# Let's say we wanted to return the string with the longest length, we specify the key as 'key = len'
print(max(word_list_3, key = len))      # ancient

    # BUT NOW, what happens if the strings all have the same length (or if more than one max is possible)?
    # the first max that appeared is the one that will be returned
word_list_4 = ['books', 'trees', 'maps']
print(max(word_list_4, key = len))          # books

word_list_5 = ['trees', 'books', 'maps']
print(max(word_list_5))                     # trees

# BE CAREFUL, uppercase letters (A-Z) come BEFORE lowercase (a-z)
word_list_6 = ['Apple', 'book', 'Book']
print(f"Max with upper and lower: {max(word_list_6)}")       # Max with upper and lower: book


# ------------------------------------------------------------------------------------------
# 2) min() -> find the smallest value in a list
```

```python
# The logic for min is the reverse of that for max() above

# float_list = [1.12, 2.54, -1.156, 3.75, 1.26]
print(min(float_list))          # -1.156

# word_list_1 = ["dog", "cat"]
print(min(word_list_1))         # cat

# word_list_2 = ['pace', 'pack']
print(min(word_list_2))         # pace

# word_list_3 = ['apple', 'ancient']
print(min(word_list_3))         # ancient

    # adding the 'len' key
print(min(word_list_3, key = len))  # apple

# ---------------------------------------------------------------------------------------
# 3) Sorting a list - Method will modify the original list

# First we consider lists with numeric values, by default the sort() method will
# sort in ascending order (smallest to biggest)

int_list = [1, 4, 9, 3, 15, 25, 8, 7, 5, 4, 1]
int_list.sort()
print(int_list)         # [1, 1, 3, 4, 4, 5, 7, 8, 9, 15, 25]

    # The sort() method has a paramater called 'reverse' set to False by default
    # False means 'sort in reverse = False' so sort in ascending order

    # if we want to sort in descending order: reverse = True

int_list.sort(reverse=True)
print(int_list)        # [25, 15, 9, 8, 7, 5, 4, 4, 3, 1, 1]


# Now consider lists of string values
# By default, the list is sorted in ascending, Alphabetical Order
groceries_list = ["coffee", "sugar", "bread", "rice", "tomato"]
groceries_list.sort()
print(groceries_list)   # ['bread', 'coffee', 'rice', 'sugar', 'tomato']

    # to specify descending order (reverse alphabetical order), set 'reverse=True
groceries_list.sort(reverse=True)
print(groceries_list)   # ['tomato', 'sugar', 'rice', 'coffee', 'bread']

    # If we want to sort by length, we specify 'key=len'
groceries_list.sort(key = len)
print(groceries_list)   # ['rice', 'sugar', 'bread', 'tomato', 'coffee']

# BE CAREFUL, lists of incomparable types cannot be sorted
# random_list = ['a', 3, 'grapes', 4.98]
# random_list.sort()            # TypeError


# ---------------------------------------------------------------------------------------
# 3) count() allows us to find the number of times a specified item appears in a list

# the list item must match the comparison item EXACTLY
letters_list = ['a', 'b', 'a', 'b', 'aa', 'a', 'bb']
print(f"count of 'a' is {letters_list.count("a")}")     # count of 'a' is 3

# ---------------------------------------------------------------------------------------
# 3) The join() method is useful in converting list items into a string and it allows us specify
#    the character to use between the joined items. This method is actually a string method, but
#    is a very helpful one to know

sentence_list = ["Welcome", "to", "a", "Python", "List", "Summary"]
sentence_string = " ".join(sentence_list)
print(sentence_string)      # Welcome to a Python List Summary

    # BE CAREFUL, this only works if the list contains strings
# dice_options = [1, 2, 3, 4, 5, 6]
# starting sentence = "The possible values for a die are: "
```

```python
# gameplay_sentence = starting_sentence + ", ".join(dice_options)     # TypeError

    # But we can use list comprehension here:
dice_options = [1, 2, 3, 4, 5, 6]
starting_sentence = "The possible values for a die are: "
# list comprehension with a TypeCase for int to string
gameplay_sentence = starting_sentence + ", ".join(str(option) for option in dice_options)
print(gameplay_sentence)    # The possible values for a die are: 1, 2, 3, 4, 5, 6

# Consider using an exclamation mark instead of a space to join:
card_suites = ['Clubs', 'Hearts', 'Spades', 'Diamonds']
card_options = "!".join(card_suites)
print(card_options)     # Clubs!Hearts!Spades!Diamonds


"""
============================================================
Copying Lists

Note that the 'copy' module needs to be imported for this section (seen at the top of the program)
"""
# If we create a list, and then attempt to copy it by using the assignment (=) operator
# we find something interesting happens:

# Let's create a list with an intentional spelling error on 'Sunday', but assign this list
# to another name before making a correction (pretend we were trying to copy it)
day_list = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Snday"]
second_day_list = day_list

day_list[6] = "Sunday"
print(day_list)             # ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
print(second_day_list)      # ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']

# I made a change to one list, but also affected the other

# The two list names are pointing to the exact same location in memory (to the same list values), so if
# we change one, we change the other (we are actually only making one change with both list names pointing to the
# same list). Here, we have not made what is called a 'True Copy'

# To make a true copy, we can use the 'copy' module, BUT there is more to consider here:
# There are two types of copy methods we can use
    # A copy that is one-level deep - this uses 'copy'
    # A copy that is all-levels deep - this uses 'deepcopy'

# Depth here refers to nested lists, but is better understood in demonstration

# 1) Consider a list that is only level deep, such as our day_list (no nested lists)
second_day_list = copy.copy(day_list)

second_day_list[0] = "First Day"
print(day_list)             # ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
print(second_day_list)      # ['First Day', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']

    # Above, changes to one list do not affect the other

# 2) Lists that have multiple levels of depth (nested lists)

# Consider an initial list that holds nested lists:
grouped_nums_list = [[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]]
second_grouped_list = copy.copy(grouped_nums_list)

# First we will change an outer list (such as the group [1, 2])
second_grouped_list[0] = [11, 12]
print(grouped_nums_list)        # [[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]]
print(second_grouped_list)      # [[11, 12], [3, 4], [5, 6], [7, 8], [9, 10]]

    # Here it appears that we have made a full copy of the original list, we can
    # see that I made a change to the second list that did not affect the first (one level deep)

# BUT what happens if I try to change an inner value
# First we will set the second_list to once again be a one-level deep copy of the first
second_grouped_list = copy.copy(grouped_nums_list)
print(second_grouped_list)      # [[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]]

```

```
# Now we will perform a change within the inner list
second_grouped_list[0][0] = 11
print(grouped_nums_list)        # [[11, 2], [3, 4], [5, 6], [7, 8], [9, 10]]
print(second_grouped_list)      # [[11, 2], [3, 4], [5, 6], [7, 8], [9, 10]]

    # NOTE: I made a change to the second list, but have now affected the first
    # The reason is because we did not make a true, complete copy of the first list
    # by only using the 'copy' method of the 'copy' module, we made a one-level deep copy

    # To make a true copy, we use the 'deepcopy' method

second_grouped_list = copy.deepcopy(grouped_nums_list)
second_grouped_list[0][0] = 12
print(grouped_nums_list)        # [[11, 2], [3, 4], [5, 6], [7, 8], [9, 10]]
print(second_grouped_list)      # [[12, 2], [3, 4], [5, 6], [7, 8], [9, 10]]

# Now we can see that when I made a deeper change to the second list (working in the nested lists), I do
# not make any changes to the original list. The second list is now a True Copy of the first
```