Chapter 5: Collections using Lists, Tuples and Dictionary

- Collections are used to store multiple items in a single variable.
- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple Set and Dictionary, all with different qualities and usage.

Python Collections

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered and changeable. No duplicate members.

1. **List**
   Lists are created using square brackets:

   Syntax

   ```
   X=[<items>]
   ```

   <items> values of a list separated by comma

   Some examples:

```
thislistoffruits = ["apple", "banana", "cherry"]
print(thislistoffruits)

thislistofnumbers = [1,2,3]
print(thislistofnumbers)
```

List Items

- List items are ordered, changeable, and allow duplicate values.
- List items are indexed, the first item has index [0], the second item has index [1] etc.

```
thislistoffruits = ["apple", "banana", "cherry"]
print(thislistoffruits[0])
print(thislistoffruits[1])
```

Ordered

- When we say that lists are ordered, it means that the items have a defined order, and that order will not change.
- If you add new items to a list, the new items will be placed at the end of the list.

Changeable

- The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

```
thislistoffruits = ["apple", "banana", "cherry"]
thislistoffruits[0]="greenapple"
print(thislistoffruits)
```

You cannot modify / add elements outside of the index number

```
thislistoffruits = ["apple", "banana", "cherry"]
thislistoffruits[3]="greenapple"
***ERROR***
```

Allow Duplicates

- Since lists are indexed, lists can have items with the same value:

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

List Length

- To determine how many items a list has, use the len() function:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

List Items - Data Types

```
list1 = ["apple", "banana", "cherry"]

list2 = [1, 5, 7, 9, 3]

list3 = [True, False, False]

list1 = ["abc", 34, True, 40, "male"]
```

type()

- From Python's perspective, lists are defined as objects with the data type 'list':

```
<class 'list'>
```

```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

The list() Constructor

- It is also possible to use the list() constructor when creating a new list.

```
thislist = list(("apple", "banana", "cherry")) # note the double round-
brackets
print(thislist)
```

Access Items

- List items are indexed and you can access them by referring to the index number

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

Negative Indexing

- Negative indexing means start from the end
- -1 refers to the last item, -2 refers to the second last item etc.

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

Range of Indexes

- You can specify a range of indexes by specifying where to start and where to end the range.
- When specifying a range, the return value will be a new list with the specified items.

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

By leaving out the start value, the range will start at the first item:

Range of Negative Indexes

- Specify negative indexes if you want to start the search from the end of the list:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1]
```

Check if Item Exists

- To determine if a specified item is present in a list use the **in** keyword:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

Change a Range of Item Values

- To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

- If you insert *more* items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

```
thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)
```

Insert Items

To insert a new list item, without replacing any of the existing values, we can use the `insert()` method.

The `insert()` method inserts an item at the specified index:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

Append Items

To add an item to the end of the list, use the `append()` method:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

Extend List

To append elements from *another list* to the current list, use the `extend()` method.

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

Remove Specified Item

The `remove()` method removes the specified item.

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

Remove Specified Index

The `pop()` method removes the specified index.

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

The del keyword also removes the specified index:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)

Delete the entire list:
del thislist
```

Clear the List

The `clear()` method empties the list.

The list still remains, but it has no content

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

Loop Through a List

You can loop through the list items by using a `for` loop

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

Loop Through the Index Numbers

You can also loop through the list items by referring to their index number.

Use the `range()` and `len()` functions to create a suitable iterable.

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

Using a While Loop

You can loop through the list items by using a `while` loop.

Use the `len()` function to determine the length of the list, then start at 0 and loop your way through the list items by refering to their indexes.

Remember to increase the index by 1 after each iteration.

```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1
```

Looping Using List Comprehension

List Comprehension offers the shortest syntax for looping through lists:

```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Example:

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

Without list comprehension you will have to write a for statement with a conditional test inside:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

print(newlist)
```

With list comprehension you can do all that with only one line of code:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

The Syntax

```
newlist = [expression for item in iterable if condition == True]
```

The return value is a new list, leaving the old list unchanged.

Condition

The *condition* is like a filter that only accepts the items that valuate to True.

```
newlist = [x for x in fruits if x != "apple"]

newlist = [x for x in range(10)]

newlist = [x for x in range(10) if x < 5]

newlist = [x.upper() for x in fruits]

newlist = ['hello' for x in fruits]
```

The *expression* can also contain conditions, not like a filter, but as a way to manipulate the outcome:

```
newlist = [x if x != "banana" else "orange" for x in fruits]
```

"Return the item if it is not banana, if it is banana return orange".

Sort List Alphanumerically

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

Sort List numerically

```
thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
```

Sort Descending

To sort descending, use the keyword argument `reverse = True`:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

Case Insensitive Sort

By default the `sort()` method is case sensitive, resulting in all capital letters being sorted before lower case letters:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
print(thislist)

# unexpected results, so use as below

thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort(key = str.lower)
print(thislist)
```

Reverse Order

What if you want to reverse the order of a list, regardless of the alphabet?

The reverse() method reverses the current sorting order of the elements.

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

Copy a List

You cannot copy a list simply by typing `list2 = list1`, because: `list2` will only be a *reference* to `list1`, and changes made in `list1` will automatically also be made in `list2`.

There are ways to make a copy, one way is to use the built-in List method `copy()`

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)

or

thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the `+` operator.

```
# Method 1
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
```

```
list3 = list1 + list2
print(list3)
# Method 2
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)

# Method 3



list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

Enumeration of lists

To find the index and the corresponding values, use enumeration

```
items=["a","b","c","d"]
for index,value in enumerate(items ):
    print(index,value)
```

List Methods

Python has a set of built-in methods that you can use on lists.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list

count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Tuple

Tuples are used to store multiple items in a single variable.

A tuple is a collection which is ordered and **unchangeable(Immutable)**.

Tuples are written with round brackets.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index `[0]`, the second item has index `[1]` etc.

Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Allow Duplicates

Since tuples are indexed, they can have items with the same value:

```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

Tuple Length

To determine how many items a tuple has, use the `len()` function:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

Create Tuple With One Item

To create a tuple with only one item, you have to add a **comma** after the item, otherwise Python will not recognize it as a tuple.

```
thistuple = ("apple",)
print(type(thistuple))
#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

Tuple Items - Data Types

Tuple items can be of any data type:

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)

A tuple can contain different data types:

tuple1 = ("abc", 34, True, 40, "male")
```

type()

From Python's perspective, tuples are defined as objects with the data type 'tuple':

```
mytuple = ("apple", "banana", "cherry")
print(type(mytuple))
```

Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

```
thistuple =("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[1])
print(thistuple[-1])
print(thistuple[2:5])
print(thistuple[:4])
print(thistuple[2:])
print(thistuple[-4:-1])

if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

Unpacking a Tuple

When we create a tuple, we normally assign values to it. This is called "packing" a tuple:
But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

Using Asterisk*

If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list:

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)
```

Loop Through a Tuple

You can loop through the tuple items by using a for loop.

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
----------------
for i in range(len(thistuple)):
  print(thistuple[i])
----------------------
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it w found

Set

Sets are used to store multiple items in a single variable.

A set is a collection which is *unordered*, *unchangeable*, and *unindexed*.
Set *items* are unchangeable, but you can remove items and add new items.

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Unchangeable

Set items are unchangeable, meaning that we cannot change the items after the set has been created.

Once a set is created, you cannot change its items, but you can remove items and add new items.

Duplicates Not Allowed

Sets cannot have two items with the same value.

Duplicate values will be ignored:

Get the Length of a Set

To determine how many items a set has, use the `len()` method.

```
thisset = {"apple", "banana", "cherry"}

print(len(thisset))
```

Set Items - Data Types

Set items can be of any data type:

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
set1 = {"abc", 34, True, 40, "male"}
print(type(set1))
```

The set() Constructor

It is also possible to use the `set()` constructor to make a set.

```
thisset = set(("apple", "banana", "cherry")) # note the double round-brackets
print(thisset)
```

Access Items

Same as List or tuple

Add Items

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

print(thisset)
```

Add Any Iterable

The object in the `update()` method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

```
thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]

thisset.update(mylist)

print(thisset)
```

Loop Items

You can loop through the set items by using a for loop:

Join Two Sets

There are several ways to join two or more sets in Python.

You can use the `union()` method that returns a new set containing all items from both sets, or the `update()` method that inserts all the items from one set into another:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)

set1.update(set2)
print(set1)
```

Keep ONLY the Duplicates

The `intersection_update()` method will keep only the items that are present in both sets.

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.intersection_update(y)

print(x)
```

The `intersection()` method will return a *new* set, that only contains the items that are present in both sets.

```
z = x.intersection(y)
```

Dictionary

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered, changeable and do not allow duplicates.

Dictionaries are written with curly brackets, and have keys and values:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Dictionary Items

Dictionary items are ordered, changeable, and does not allow duplicates.

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

Changeable

Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

Duplicates Not Allowed

Dictionaries cannot have two items with the same key:

Duplicate values will overwrite existing values:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)
```

Dictionary Length

To determine how many items a dictionary has, use the `len()` function:

Dictionary Items - Data Types

The values in dictionary items can be of any data type:

```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
```

```
print(type(thisdict))
```

Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car ["model"]

x = car.get("model")

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change

x = thisdict.values(). # Get the values

print(x) #before the change

car["year"] = 2020

print(x) #after the change
```

The `items()` method will return each item in a dictionary, as tuples in a list.

```
x = car.items()

if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Change Values

You can change the value of a specific item by referring to its key name:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
```

```
thisdict.update({"year": 2020})
```

Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)

thisdict.update({"color": "red"})
```

Loop Through a Dictionary

You can loop through a dictionary by using a `for` loop.

When looping through a dictionary, the return value are the *keys* of the dictionary, but there are methods to return the *values* as well.

```
for x in thisdict:
  print(x) or print(thisdict[x])

for x in thisdict.values():
  print(x)

for x in thisdict.keys():
  print(x)
```

Nested Dictionaries

A dictionary can contain dictionaries, this is called nested dictionaries.

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
```

```
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
```

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

Method	Description
clear	Removes all the elements from the dictionary
copy	Returns a copy of the dictionary

fromkeys	Returns a dictionary with the specified keys and value
get	Returns the value of the specified key
items	Returns a list containing a tuple for each key value pair
keys	Returns a list containing the dictionary's keys
pop	Removes the element with the specified key
popitem	Removes the last inserted key-value pair
setdefault	Returns the value of the specified key. If the key does not exist: insert the key, with specified value
update	Updates the dictionary with the specified key-value pairs
values	Returns a list of all the values in the dictionary

Problems

Problem : Write a Python program to sum all the items in a list.

Problem : Write a Python program to multiply all the items in a list.

Problem : Write a Python program to get the largest number from a list.(Use sort())

Problem : Write a Python program to count the number of strings where the string length is 2 or more and the first and last character are same from a given list of strings.

Problem : Write a Python program to check a list is empty or not.

Problem : Write a Python program to clone or copy a list.

Problem : Write a Python program to clone or copy a list.

Problem : Write a Python program to find the list of words that are longer than n(3) from a given list of words
Sample input: `"The quick brown fox jumps over the lazy dog"`
Output : `['quick', 'brown', 'jumps', 'over', 'lazy']`
(Prints all the words whose length is greater 3)

Problem 10: Write a Python program to print the numbers of a specified list after removing even numbers from it.(use with and without list comprehension)

Problem 11: Write a Python program to shuffle and print a specified list.

Hint: use `from random import shuffle`

Problem 12: Write a Python program to access the index of a list.(Use enumerate)

Problem 13: Write a Python program to print a specified list after removing the 0th, 4th and 5th elements.

Problem 14: Write a Python program to print a specified list after removing the 0th, 4th and 5th elements.

Problem 15: Write a Python program to find the index of an item in a specified list.

Problem 16: Write a Python program to append a list to the second list.

Problem 18: Python program to display the sum of n numbers using a list.

Sample Input: User enters `How many numbers`: 5

System should ask use to enter upto 5 numbers and then it should sum all the numbers and give output

Problem 19: Python program to find the odd numbers in an array

Problem 20: Python program to delete an element from a list by index

Problem 21: Python program to print all the items in a dictionary as key:value

```
phone_book = {
'John' : [ '8592970000', 'john@xyzmail.com' ],
'Bob': [ '7994880000', 'bob@xyzmail.com' ],
'Tom' : [ '9749552647' , 'tom@xyzmail.com' ]
}
```

Problem 22: Write a Python program which accepts a sequence of comma-separated numbers from user and generate a list and a tuple with those numbers
Sample data : 3, 5, 7, 23
Output :
List : ['3', ' 5', ' 7', ' 23']
Tuple : ('3', ' 5', ' 7', ' 23')

Problem 23: Write a Python program that accepts a comma separated sequence of words as input and prints the unique words in sorted form (alphanumerically).

Problem 24: Write a Python program to count and display the vowels of a given text.

Problem 25: Write a Python program to convert a given string into a list of words.

Problem 27: Write a Python program to remove leading zeros from an IP address.

Input : ”10.001.002.012”
Output: ”10.1.2.12”

Problem 28: Write a Python program to split a given multiline string into a list of lines
Input: '"This
is a
multiline
string"'

Problem 29: Write a Python program to count the number of even and odd numbers from a series of numbers from the given input
Sample numbers : numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9)
Expected Output :
Odd numbers : [1,3,5,7,9](5)
Even numbers : [2,4,6,8](4)

Problem 30: Write a Python program which accepts a sequence of comma separated 4 digit binary numbers as its input and print the numbers that are divisible by 5 in a comma separated sequence.

Problem 31: Write a Python program to find numbers between 100 and 400 (both included) where each digit of a number is an even number. The numbers obtained should be printed in a comma-separated sequence.

Problem 32: Write a Python program to concatenate elements of a list.

Problem 34: Write a Python program using List Comprehend to create a new list

from a given input list. The new list should contain only those elements that start with a specific character

Ex: `lst = ["abcd", "abc", "bcd", "bkie", "cder", "cdsw", "sdfsd", "dagfa", "acjd"]`
Print a new list with all the elements that start with “a”

Problem 35: Write a Python program to remove the K'th element from a given list, print the new list.

```
n_list = [1,1,2,3,4,4,5,1]
Kth position = 3
```

Problem 36: Write a Python program to accept a fruit name from input repeatedly until the user enters Exit or exit, and displays all the fruits entered in a list

Problem 37: Accept a word from input and print it’s meaning. Shows all the words to the user for which the meaning is to be displayed. Use a dictionary with a word and meaning

Input:

mydict={

"abate":" become less intense or widespread.",

"absurd":"stupid and unreasonable, or silly in a humorous way",

"potential":"possible when the necessary conditions exist:",

"appeal":"a request to the public for money, information, or help"

}

Output:

The meaning of abate is “become less intense or widespread”

Problem 38: Write a Python program that accepts a sequence of lines (blank line to terminate) as input and prints the lines as output (all characters in upper case as output).

Problem : Write a Python program to convert month name to a number of days.

User enter a month name
System outputs : no of days in that month.