Advanced Data Structures / Data Types

List:

- List is the mutable sequences, which is often used to store the data of same type.
- A list in Python is known as a "sequence data type" like strings. It is an ordered collection of values enclosed within square brackets [ ].
- Each value of a list is called as element. It can be of any type such as numbers, characters, strings and even the nested lists as well.
- The elements can be modified or mutable which means the elements can be replaced, added or removed. Every element rest at some position in the list.
- The position of an element is indexed with numbers beginning with zero which is used to locate and access a particular element.
    - A list is a container object
    - Heterogeneous sequence of elements
    - Can have duplicates
    - Mutable
    - Elements are separated by comma, enclosed in [ ] parenthesis

List Methods:

Function	Description
Append()	Add an element to the end of the list
Extend()	Add all elements of a list to another list
Insert()	Insert an item at the defined index
Remove()	Removes an item from the list
Pop()	Removes and returns an element at the given index
Clear()	Removes all items from the list
Index()	Returns the index of the first matched item
Count()	Returns the count of number of items passed as an argument
Sort()	Sort items in a list in ascending order
Reverse()	Reverse the order of items in the list
copy()	Returns a copy of the list

Built-in functions with List:

Function	Description

reduce()	apply a particular function passed in its argument to all of the list elements stores the intermediate result and only returns the final summation value
sum()	Sums up the numbers in the list
ord()	Returns an integer representing the Unicode code point of the given Unicode character
cmp()	This function returns 1, if first list is “greater” than second list
max()	return maximum element of given list
min()	return minimum element of given list
all()	Returns true if all element are true or if list is empty
any()	return true if any element of the list is true. if list is empty, return false
len()	Returns length of the list or size of the list
enumerate()	Returns enumerate object of list
accumulate()	apply a particular function passed in its argument to all of the list elements returns a list containing the intermediate results
filter()	tests if each element of a list true or not
map()	returns a list of the results after applying the given function to each item of a given iterable
lambda()	This function can have any number of arguments but only one expression, which is evaluated and returned.

Tuple:

- Tuples consists of a number of values separated by comma and enclosed within parentheses.
- Tuple is similar to list, values in a list can be changed but not in a tuple.
- Tuple is immutable, a tuple needs lesser memory than lists.
    - Another type of container object
    - Heterogeneous sequence of elements
    - Can have duplicates
    - Immutable
    - Elements are separated by comma, enclosed in ( ) parenthesis

Advantages of Tuples over list:

- The elements of a list are changeable (mutable) whereas the elements of a tuple are unchangeable (immutable), this is the key difference between tuples and list.
- The elements of a list are enclosed within square brackets. But, the elements of a tuple are enclosed by parenthesis.
- Iterating tuples is faster than list.

Built-In Methods:

Built-in Function	Description
all()	Returns true if all element are true or if tuple is empty
any()	return true if any element of the tuple is true. if tuple is empty, return false
len()	Returns length of the tuple or size of the tuple
enumerate()	Returns enumerate object of tuple
max()	return maximum element of given tuple
min()	return minimum element of given tuple
sum()	Sums up the numbers in the tuple
sorted()	input elements in the tuple and return a new sorted list
tuple()	Convert an iterable to a tuple.

Dictionary:

- A dictionary is a mixed collection of elements. Unlike other collection data types such as a list or tuple, the dictionary type stores a key along with its element.
- The keys in a Python dictionary is separated by a colon (:) while the commas work as a separator for the elements. The key value pairs are enclosed with curly braces { }.
- Key in the dictionary must be unique case sensitive and can be of any valid Python type
    - A Python dictionary is a mapping of unique keys to values
    - Use {} curly brackets to construct the dictionary
    - [] square brackets to index it
    - Dictionaries are mutable

Difference between List and Dictionary:

(1) List is an ordered set of elements. But a dictionary is a data structure that is used for matching one element (Key) with another (Value).

(2) The index values can be used to access a particular element. But, in dictionary key represents index. Remember that, key may be a number of a string.

(3) Lists are used to look up a value whereas a dictionary is used to take one value and look up another value.

Dictionary Methods:

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 the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Set:

- A Set is a mutable and an unordered collection of elements without duplicates.
- That means the elements within a set cannot be repeated.
- This feature used to include membership testing and eliminating duplicate elements.
    - Sets are very similar to list data structures
    - Do not allow duplicates
    - Unordered collections of homogeneous elements
    - Sets are generally used to remove duplicate elements from a list

Set Methods:

Function	Description
add()	Adds an element to a set
remove()	Removes an element from a set. If the element is not present in the set, raise a KeyError
clear()	Removes all elements form a set
copy()	Returns a shallow copy of a set
pop()	Removes and returns an arbitrary set element. Raise KeyError if the set is empty
update()	Updates a set with the union of itself and others

union()	Returns the union of sets in a new set
difference()	Returns the difference of two or more sets as a new set
difference_update()	Removes all elements of another set from this set
discard()	Removes an element from set if it is a member. (Do nothing if the element is not in set)
intersection()	Returns the intersection of two sets as a new set
intersection_update()	Updates the set with the intersection of itself and another
isdisjoint()	Returns True if two sets have a null intersection
issubset()	Returns True if another set contains this set
issuperset()	Returns True if this set contains another set
symmetric_difference()	Returns the symmetric difference of two sets as a new set
symmetric_difference_update()	Updates a set with the symmetric difference of itself and another

Sample Interview Questions:

1. What is the difference between a list and a tuple?
2. What is the difference between an array and a list?
3. What is meant by a dictionary in Python?
4. What's a negative index?
5. What are mutable and immutable data types?
6. What is the difference between tuple and dictionary?

Map Functions

- map() is a built-in function that allows you to process and transform all the items in an iterable without using an explicit for loop, a technique commonly known as mapping.
- map() is useful when you need to apply a transformation function to each item in an iterable and transform them into a new iterable.
- map() is one of the tools that support a functional programming style in Python.
- Mapping consists of applying a transformation function to an iterable to produce a new iterable. Items in the new iterable are produced by calling the transformation function on each item in the original iterable.

Understanding map():

- map() loops over the items of an input iterable (or iterables) and returns an iterator that results from applying a transformation function to every item in the original input iterable.
- According to the documentation, map() takes a function object and an iterable (or multiple iterables) as arguments and returns an iterator that yields transformed items on demand. The function's signature is defined as follows:

map(function, iterable[, iterable1, iterable2,..., iterableN])

- map() applies function to each item in iterable in a loop and returns a new iterator that yields transformed items on demand.
- function can be any Python function that takes a number of arguments equal to the number of iterables you pass to map().
- This first argument to map() is a transformation function. In other words, it's the function that transforms each original item into a new (transformed) item.
- Even though the Python documentation calls this argument function, it can be any Python callable. This includes built-in functions, classes, methods, lambda functions, and user-defined functions.
- The operation that map() performs is commonly known as a mapping because it maps every item in an input iterable to a new item in a resulting iterable.
- To do that, map() applies a transformation function to all the items in the input iterable.

Examples:

1)

```
def myfunc(a, b):
    return a + b

x = map(myfunc, ('apple', 'banana', 'cherry'), ('orange', 'lemon', 'pineapple'))
print(list(x))

>> ['appleorange', 'bananalemon', 'cherrypineapple']
```

2)

```
def powers(x):
    return x ** 2, x ** 3
```

```
numbers = [1, 2, 3, 4]
print(list(map(powers, numbers)))

>> [(1, 1), (4, 8), (9, 27), (16, 64)]
```

3)

```
import math
numbers = [1, 2, 3, 4, 5, 6, 7]

print(list(map(math.factorial, numbers)))

>> [1, 2, 6, 24, 120, 720, 5040]
```

4)

```
numbers = [ 74, 85, 14, 23, 56, 31,44 ]
remainders = map(lambda num: num%5, numbers)
for i in remainders:
   print(i)
```

5)

```
first_it = [1, 2, 3]
second_it = [4, 5, 6, 7]

print(list(map(pow, first_it, second_it)))

>> [1, 32, 729]
```

6)

```
list(map(lambda x, y, z: x + y + z, [2, 4], [1, 3], [7, 8]))

>> [10, 15]
```

7)

```
string_it = ["processing", "strings", "with", "map"]
list(map(str.capitalize, string_it))

>> ['Processing', 'Strings', 'With', 'Map']
```

8)
```
def myMapFunc(list1, tuple1):
   return list1+"_"+tuple1

my_list = ['a','b', 'b', 'd', 'e']
my_tuple = ('PHP','Java','Python','C++','C')

updated_list = map(myMapFunc, my_list,my_tuple)
print(updated_list)
print(list(updated_list))

>> ['a_PHP', 'b_Java', 'b_Python', 'd_C++', 'e_C']
```

Python String Built-In Functions

Python String capitalize()	Converts first character to Capital Letter
Python String casefold()	converts to case folded strings
Python String center()	Pads string with specified character
Python String count()	returns occurrences of substring in string
Python String encode()	returns encoded string of given string
Python String endswith()	Checks if String Ends with the Specified Suffix
Python String expandtabs()	Replaces Tab character With Spaces
Python String find()	Returns the index of first occurrence of substring
Python String format()	formats string into nicer output
Python String format_map()	Formats the String Using Dictionary
Python String index()	Returns Index of Substring
Python String isalnum()	Checks Alphanumeric Character
Python String isalpha()	Checks if All Characters are Alphabets
Python String isdecimal()	Checks Decimal Characters
Python String isdigit()	Checks Digit Characters

Python String isidentifier()	Checks for Valid Identifier
Python String islower()	Checks if all Alphabets in a String are Lowercase
Python String isnumeric()	Checks Numeric Characters
Python String isprintable()	Checks Printable Character
Python String isspace()	Checks Whitespace Characters
Python String istitle()	Checks for Title Case String
Python String isupper()	returns if all characters are uppercase characters
Python String join()	Returns a Concatenated String
Python String ljust()	returns left-justified string of given width
Python String lower()	returns lowercase string
Python String lstrip()	Removes Leading Characters
Python String maketrans()	returns a translation table
Python String partition()	Returns a Tuple

Python String replace()	Replaces Substring Inside
Python String rfind()	Returns the Highest Index of Substring
Python String rindex()	Returns Highest Index of Substring
Python String rjust()	returns right-justified string of given width
Python String rpartition()	Returns a Tuple
Python String rsplit()	Splits String From Right
Python String rstrip()	Removes Trailing Characters
Python String split()	Splits String from Left
Python String splitlines()	Splits String at Line Boundaries
Python String startswith()	Checks if String Starts with the Specified String
Python String strip()	Removes Both Leading and Trailing Characters
Python String swapcase()	swap uppercase characters to lowercase; vice versa
Python String title()	Returns a Title Cased String
Python String translate()	returns mapped charactered string
Python String upper()	returns uppercased string
Python String zfill()	Returns a Copy of The String Padded With Zeros

Python – Predefined Functions

Built In function	Purpose
Python abs()	returns absolute value of a number
Python all()	returns true when all elements in iterable is true
Python any()	Checks if any Element of an Iterable is True
Python ascii()	Returns String Containing Printable Representation
Python bin()	converts integer to binary string
Python bool()	Converts a Value to Boolean
Python bytearray()	returns array of given byte size
Python bytes()	returns immutable bytes object
Python callable()	Checks if the Object is Callable

Python chr()	Returns a Character (a string) from an Integer
Python classmethod()	returns class method for given function
Python compile()	Returns a Python code object
Python complex()	Creates a Complex Number
Python delattr()	Deletes Attribute From the Object
Python dict()	Creates a Dictionary
Python dir()	Tries to Return Attributes of Object
Python divmod()	Returns a Tuple of Quotient and Remainder
Python enumerate()	Returns an Enumerate Object
Python eval()	Runs Python Code Within Program
Python exec()	Executes Dynamically Created Program
Python filter()	constructs iterator from elements which are true
Python float()	returns floating point number from number, string
Python format()	returns formatted representation of a value
Python frozenset()	returns immutable frozenset object
Python getattr()	returns value of named attribute of an object
Python globals()	returns dictionary of current global symbol table
Python hasattr()	returns whether object has named attribute

Python hash()	returns hash value of an object
Python help()	Invokes the built-in Help System
Python hex()	Converts to Integer to Hexadecimal
Python id()	Returns Identify of an Object
Python input()	reads and returns a line of string
Python int()	returns integer from a number or string
Python isinstance()	Checks if a Object is an Instance of Class
Python issubclass()	Checks if a Class is Subclass of another Class
Python iter()	returns an iterator
Python len()	Returns Length of an Object
Python list()	creates a list in Python

Python locals()	Returns dictionary of a current local symbol table
Python map()	Applies Function and Returns a List
Python max()	returns the largest item
Python memoryview()	returns memory view of an argument
Python min()	returns the smallest value
Python next()	Retrieves next item from the iterator
Python object()	creates a featureless object
Python oct()	returns the octal representation of an integer
Python open()	Returns a file object
Python ord()	returns an integer of the Unicode character
Python pow()	returns the power of a number
Python print()	Prints the Given Object
Python property()	returns the property attribute
Python range()	return sequence of integers between start and stop
Python repr()	returns a printable representation of the object
Python reversed()	returns the reversed iterator of a sequence
Python round()	rounds a number to specified decimals

Python set()	constructs and returns a set
Python setattr()	sets the value of an attribute of an object
Python slice()	returns a slice object
Python sorted()	returns a sorted list from the given iterable
Python staticmethod()	transforms a method into a static method
Python str()	returns the string version of the object
Python sum()	Adds items of an Iterable
Python super()	Returns a proxy object of the base class
Python tuple()	Returns a tuple
Python type()	Returns the type of the object
Python vars()	Returns the __dict___attribute
Python zip()	Returns an iterator of tuples
Python_import_()	Function called by the import statement