Core Data Structures In Python

WHAT IS DATA STRUCTURE

Hello, students. Welcome to Core Data Structures, a fresh and exciting module. We've come a long way in our learning of the Python programming language.
You will be learning about core data structures like List, Tuple, Dictionary, Sets, Frozensets in this module.

Let us start with what is data structure? and its applications.

Data is an important aspect when it comes to programming.
Data Structure is a primary memory allocation to store, organize and manage data for efficient access and modifications.
The essential constructs that support the development of your programs are data structures.
Each data structure provides a unique way of organizing data so that it may be accessed fast, depending on your use case.
Let's look at it through the lens of an analogy.
Consider how your workspace could be organised to meet your specific needs, allowing you to work more efficiently with easy access to all of your tools.
The same is true for data; it must be stored efficiently to allow for easier access and change. Data structure refers to the organization and processing of the data in primary memory storage.

Let's have a look at the many forms of data structures.
There are two types of data structures in Python.
The list, tuple, set, dictionary, and frozen set are among the built-in data structures.
The second type, known as user defined data structure, namely Linked lists, stacks, queues, trees, graphs, and hash maps are among them can be created using Python.

The built-in data structure, such as List, Tuple, Set, Dictionary, Frozen sets are represented technically using the terminologies such as sequence, container and collection.
Sequence can hold several data types at the same time such as integers, strings, complex numbers.
Container is like a variable, it can hold data of various types all at the same time.
Collection there is one more type called collection data type. This can hold many data types.

Moving on to Array. What is an array?
A Python Array is a collection of common types of data structures having elements with the same data type. It is used to store collections of data. Arrays are static in nature. It can be resized, you can add elements and do declarations.

Let us look at the example of an array. In this example you can import data in the array and declare it.

Let's take a look at a useful built-in data structure called a list.

Lists are similar to dynamic arrays in Python.

A dynamic array is able to resize and add elements after declaration. In Python, a list is a dynamic array.

The list can contain data of different types.

The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

It uses a constructor to create a list.

constructor list[]

A constructor is a special kind of method that Python calls when it instantiates an object using the definitions found in your class.
In Python programming, a list is created by placing all the items (elements) inside square brackets [], separated by commas.
Look at the examples

Example-1

```
#creating a list using list() #constructor

text = "Python"

print(list(text))
```

Output

```
['P', 'y', 't', 'h', 'o', 'n']
```

Example - 2

```
# empty list creation

my_list = []
```

```
# list of integers
my_list = [10, 20, 30]
print(my_list)
# list with mixed data types
my_list1 = [2021, "Python", 88.9]
print(my_list1)
```

Output

```
[10, 20, 30]
[2021, 'Python', 88.9]
```

Now let’s look at slicing operators.

With this operator, one can specify where to start the slicing, where to end, and specify the step.

This is the syntax of the slicing operator.

Lst[Initial : End : Increment or Update]

In this example

```
# Initialize list
Lst = [20, 40, 50, 20, 100, 10, 510]
 # Display list
print(Lst[1:5])
```

Index -1 represents the last element and -n represents the first element of the list

Output

```
[40, 50, 20, 100]
```

```
# Code for accessing the elements from the  last using negative indexing
Lst = [20, 40, 50, 20, 100, 10, 510]
 # Display list
print(Lst[-1:-5:-1])
```

Output

```
[510, 10, 100, 20]
```

In this example the slicing operator defines the beginning and end.

The value of the index in the list starts with zero.

list1 = [1, "hi", "python", 2] #list index starts with 0

The output will be 2

print ((list1[3:])) #Slicing Operator start:stop

The output will be 2 since there are no more elements in the list.

print ((list1[0:2])) #slicing operator will get executed stop-1 (0 to 1)

The output will be 1, hi because the index starts from zero.

print ((list1)) #prints the entire list

The out will print the entire list

print (list1 + list1) #Concatenation

The output will repeat the list

print (list(list1 * 3)) #Repetition & type conversion

Before moving further you need to understand the difference between methods and functions.

Python is object oriented programming it supports, class, objects and methods.

Methods are associated with the objects of the class they belong to. Functions are not associated with any object.

Methods are associated with an object.

We call a method on an object using the dot operator.

Whereas function is a group of statements capable of performing some tasks.

A function in a program can take arguments (that is, the data to operate on) and can optionally return some data after performing the intended task.

Let us go through the methods and functions of the list.

Method	Meaning

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
extend()	Add the elements of a list (or any iterable), to the end of the current list
count()	Returns the number of elements with the specified value

Method	Meaning
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 first item with the specified value
sort()	Sorts the list
reverse()	Reverses the order of the list

Method	Meaning
max()	return maximum element of given list
min()	return minimum element of given list
len()	Returns length of the list or size of the list

concatenate(+)	Merges two lists and returns a single list.
repetition/ iteration(*)	Allows multiplying the list n times. The resultant list is the original list is iterated n times.

I hope the concept of list is easier to understand with the methods and functions that can be used to modify the data structure.

TUPLE

Hello, and welcome back to the tuple data structure module. You'll learn about tuple operations, as well as the methods and functions that are included in it.
Let us start with what is tuple?
Tuples are used to store multiple items in a single variable. A tuple is a collection which is ordered and unchangeable or immutable.

- Tuples are written with parentheses.
- Since tuples are indexed, they can have items with the same value
- Tuples are ordered, it means that the items have a defined order, and that order will not change.
- A tuple can contain different data types
- It is also possible to use the tuple() constructor to make a tuple.

Let's begin by creating a tuple with only one element. You must include a comma after the single item, or Python will not recognize it as a tuple.

Look at the example,
thistuple = ("apple",)
print(type(thistuple))
The output is class tuple. Python will give the class.
In the second example
thistuple = tuple(("apple", "banana", "cherry")) #note the double round-brackets
print(thistuple)

The output ('apple', 'banana', 'cherry'). Here the compiler printed the elements present in the tuple.

Let's have a look at the different tuple operations.

Operations	Description	Examples
Creating a tuple	Creating the tuple with elements of different data types.	a =(20,40,60,"apple","ball")
Indexing	Accessing the ite1n in the position 0 Accessing the item in the position 2	print(a[0]) 20 a [2] 60
Slicing	Displaying items from 1st till 2nd.	print(a[1:3]) (40,60)

Concatenation	Adding tuple elements at the end of another tuple elements	b=(2,4)
		print(a+b)
		(20,40,60,"apple","ball",2,4)

Repetition	Repeating the tuple in n no of times	print(b*2) (2,4,2,4)

Operations	Description	Examples
Membership	Returns True if an element is present in the tuple. Otherwise returns false.	a=(2,3,4,5,6,7,8,9,10) 5 in a True 100 in a False 2 not in a False
Comparison	Returns True if all elements in both elements are the same.	a=(2,3,4,5,6,7,8,9,10) b=(2,3,4) a==b False a!=b True

Here are the tuple methods and functions

Methods/Functions	Description	Example

a.index(tuple)	Returns the index of the first matched item.	a=(l,2,3,4,5)
		
		a.index(5)
		4

a.count(tuple)	Returns the count of the given element.	a=(l,2,3,4,5)
		
		a.count(3)
		1

len(tuple)	return the length of the tuple	len(a)5

Methods/Functions	Description	Example
min(tuple)	return the minimum element in a tuple	min (a) 1

max(tuple)	return the maximum element in a tuple	max(a)5
del(tuple)	Delete the entire tuple.	del(a)

Here is an example for Tuple method
#Counting the number of elements in a tuple
a=(3,2,1,3,4,5)
a.count(3)
The output is 2 Count method returns the count of the elements in a tuple.

I hope you enjoyed this tuple module and that you will continue your journey to learn more about data structures. Look forward to seeing you in the upcoming video.

SET

Welcome Back to this module, you will learn about sets and set methods, as well as their advantages.

Let us start with what is set?
Sets are used to store multiple items in a single variable.
A set is a collection which is both unordered and unindexed.
Set items are unordered, changeable, and do not allow duplicate values.
Sets are written with curly brackets.
It's imperative to understand the importance of sets before you learn about its methods.
The main advantage of utilising a set over a list is that it has a highly optimised way for detecting whether a certain member is in the set. This is based on a hash table, which is a type of data structure. We can't use indexes to access items in sets because they're not arranged like lists.

I hope you got the benefits of sets as well as how they differ from lists.

Here have a look at the set methods.

Method	Meaning
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set

Method	Meaning
intersection()	Returns a set, that is the intersection of two or more sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not

Method	Meaning
symmetric_difference()	Returns a set with the symmetric differences of two sets

symmetric_difference_update()	inserts the symmetric differences from this set and another

Lets is understand better with these examples

```
A = {1, 2, 3, 4, 5}
print(A)
```

Output

```
{1, 2, 3, 4, 5}
```

The output prints all the elements of the set.
Here the set will eliminate the duplicates.

```
#Set Eliminates the duplication
A = {4, 5 , 1, 2, 3, 4, 5}
print(A)
```

Output

```
{1, 2, 3, 4, 5}
```

Well, the topic set was simple to follow with the examples, and I hope to see you in the next module.

DICTIONARY

Hello learners. We have been discussing the built-in data structures and in this module you will learn about dictionaries and operations and methods related to it.
First, what is a dictionary?
Dictionaries are written with curly brackets, and have keys and values: Dictionaries are used to store data values in key:value pairs. A dictionary is a collection which is ordered, changeable and does not allow duplicates.

As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.

Values in a dictionary can be of any datatype and can be duplicated, whereas keys can't be repeated and must be immutable.

JavaScript Object Notation: JSON is probably nowadays' mostly used model for data exchange over the internet. Mutable, Dynamic. They can grow and shrink as needed. Nested. A list can contain another list. A dictionary can contain another dictionary. A dictionary can also contain a list, and vice versa.

The basic difference between list and dictionary is that list elements are accessed by their position in the list, via indexing. Dictionary elements are accessed via keys.

Look at the example

```
#Program for personal details
P_D = {“Name”:”Karan”, “Age”:24, “City”:”Delhi”, “Height”: 5.8}
```

Output

```
{'Name': 'Karan', 'Age': 24, 'City': 'Delhi', 'Height': 5.8}
```

Let us look at the dictionary methods

Method	Meaning
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
values()	Returns a list of all the values in the dictionary
get()	Returns the value of the specified key

Method	Meaning
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
update()	Updates the dictionary with the specified key-value pairs

Let us understand the methods with these examples.

```
dict = {'Name': 'Jenifer', 'Age': 7, 'Class': 'Second'}
print(dict)
{'Name': 'Jenifer', 'Age': 7, 'Class': 'Second'}

dict['Age'] = 8; # update existing entry
print(dict)
{'Name': 'Jenifer', 'Age': 8, 'Class': 'Second'}

dict['School'] = "DPS School"; # Add new entry
print(dict)
{'Name': 'Jenifer', 'Age': 8, 'Class': 'Second', 'School': 'DPS School'}

del dict['Name']; # remove entry with key 'Name'
print(dict)
{'Age': 8, 'Class': 'Second', 'School': 'DPS School'}

dict.clear();     # remove all entries in dict
print(dict)
{}

del dict ;       # delete entire dictionary'
print(dict)
<class 'dict'>
```

Now let us look at the dictionary functions.

Function	Meaning
cmp(dict1, dict2)	Compares elements of both dict.
len(dict)	Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
str(dict)	Produces a printable string representation of a dictionary
type(variable)	Returns the type of the passed variable. If the passed variable is a dictionary, then it would return a dictionary type.

I hope this module on dictionary and its methods is clear to you.

FROZEN SET

Welcome back learners. It's a great joy to see you here again. In this module you will learn about frozen sets and the operations related to it with examples.

Let's get started with the definition.
A frozen set freezes the list, and makes it unchangeable.
The frozenset() function returns an unchangeable frozenset object (which is like a set object, only unchangeable).

Example
```
mylist = ['Python', 'Java', 'Julia']
x = frozenset(mylist)
```
Output
```
frozenset({'Julia', 'Java', 'Python'})
```

If you try to add an element to a frozen set, you will get an error.
For example

```
# tuple of vowels
vowels = ('a', 'e', 'i', 'o', 'u')
fSet = frozenset(vowels)
print('The frozen set is:', fSet)
print('The empty frozen set is:', frozenset())
```

Output

```
The frozen set is: frozenset({'e', 'i', 'a', 'o', 'u'})
The empty frozen set is: frozenset()
```

```
# frozensets are immutable
fSet.add('v')
```

If you try to add an element to a frozen set, you will get an error.

It's interesting to note that like normal sets, frozenset can also perform different operations like copy, difference, intersection, symmetric_difference, and union
Similarly, other set methods like isdisjoint, issubset, and issuperset are also available.
Check this example to see how you can copy using the frozen set operation.

```
# Frozensets
# initialize A and B
A = frozenset([1, 2, 3, 4])
B = frozenset([3, 4, 5, 6])

# copying a frozenset
C = A.copy()  # Output: frozenset({1, 2, 3, 4})
print(C)
```

Output

```
frozenset({1, 2, 3, 4})
```

Here you can combine the frozen sets by omitting the repeated elements.

```
# union
print(A.union(B))
```

Output

```
frozenset({1, 2, 3, 4, 5, 6})
```

Some more examples of the frozen set operations

```
# intersection
print(A.intersection(B))
Output
frozenset({3, 4})


# difference
print(A.difference(B))
Output
frozenset({1, 2,})


# symmetric_difference
print(A.symmetric_difference(B
Output
frozenset({1, 2, 5, 6})


# Frozensets
# initialize A, B and C
A = frozenset([1, 2, 3, 4])
B = frozenset([3, 4, 5, 6])
C = frozenset([5, 6])

# isdisjoint() method
print(A.isdisjoint(C))
Output
True

# issubset() method

print(C.issubset(B))

Output
True

# issuperset() method

print(B.issuperset(C))

Output
True
```

I hope you found this module useful.