**Section 4 Lesson 20: Understanding Python Lists**

**Introduction:**  
In this lesson, we will analyze a piece of Python code that demonstrates the creation and manipulation of lists. Lists are one of the most versatile data structures in Python, allowing us to store multiple items in a single variable. We will break down the code line by line, explaining each part and its purpose.

**Code Breakdown:**

1. **Line 1: List Initialization**
   * **Code:** nums = [1,2,3,4,5,6,7]
   * **Explanation:** This line initializes a list named nums that contains integers from 1 to 7.
   * **Purpose:** Lists are used to store multiple values in a single variable, making data management easier.
   * **Example:** If we change the list to nums = [1, 2, 3, 4, 5], the nums list will now contain only these five integers.
2. **Line 2: Print Statement**
   * **Code:** print(nums)
   * **Explanation:** This line outputs the contents of the nums list to the console.
   * **Purpose:** The print() function is used to display information to the user, allowing us to see the contents of our list.
   * **Example:** If we run this line, it will output [1, 2, 3, 4, 5, 6, 7].
3. **Line 3: List of Strings**
   * **Code:** names = ["ahmed","jermaine","mycle","mohamed"]
   * **Explanation:** This line creates a list named names that contains four string elements.
   * **Purpose:** Lists can hold items of different data types, such as strings, integers, and more. Here, we are storing names.
   * **Example:** If we change the list to names = ["Alice", "Bob"], it will now only contain these two names.
4. **Line 4: Print Statement**
   * **Code:** print(names)
   * **Explanation:** This line prints the contents of the names list to the console.
   * **Purpose:** To allow us to see what names are currently stored in the list.
   * **Example:** Running this line will output ["ahmed", "jermaine", "mycle", "mohamed"].
5. **Line 5: Mixed Data Types in a List**
   * **Code:** l1 = ["sara", 12, "ahmed", True, 5.24]
   * **Explanation:** This line initializes a list named l1 containing mixed data types: strings, an integer, a boolean, and a float.
   * **Purpose:** Lists can store different types of data, allowing for flexibility in data management.
   * **Example:** If we add another item like l1.append("new\_item"), it will now include "new\_item" in the list.
6. **Line 6: Print Statement**
   * **Code:** print(l1)
   * **Explanation:** This line prints the contents of the l1 list to the console.
   * **Purpose:** To provide visibility of the mixed data types stored in the list.
   * **Example:** The output will be ["sara", 12, "ahmed", True, 5.24].
7. **Line 7: Fruits List Initialization**
   * **Code:** fruits = ["banana", "apple", "mango", "grapes"]
   * **Explanation:** This line creates a list named fruits containing the names of four different fruits.
   * **Purpose:** To demonstrate how to create a list specifically for storing fruit names.
   * **Example:** If we change the list to fruits = ["orange", "kiwi"], it will now only contain these two fruits.
8. **Line 8: Print Statement**
   * **Code:** print(fruits)
   * **Explanation:** This line prints the contents of the fruits list to the console.
   * **Purpose:** To allow us to see the fruit names currently stored in the list.
   * **Example:** Running this line will output ["banana", "apple", "mango", "grapes"].
9. **Line 9: Accessing List Values**
   * **Code:** print(fruits[1])
   * **Explanation:** This line accesses the second element in the fruits list (index 1, since indexing starts at 0).
   * **Purpose:** To demonstrate how to retrieve specific items from a list using their index.
   * **Example:** In this case, the output will be apple.
10. **Line 10: Length of the List**
    * **Code:** print(len(fruits))
    * **Explanation:** This line prints the length of the fruits list, which is the number of items it contains.
    * **Purpose:** To show how to determine the size of a list.
    * **Example:** If fruits contains four items, the output will be 4.
11. **Line 11: List Constructor**
    * **Code:** l2 = list((12, 144, "yousef", False))
    * **Explanation:** This line creates a list named l2 using the list() constructor with a tuple as an argument.
    * **Purpose:** To illustrate how to create a list from other iterable data types.
    * **Example:** The output of print(l2) will yield [12, 144, "yousef", False].
12. **Line 12: Print Statement**
    * **Code:** print(l2)
    * **Explanation:** This line prints the contents of the l2 list to the console.
    * **Purpose:** To provide visibility of the items created in the l2 list.
    * **Example:** The output will be [12, 144, "yousef", False].

**Conclusion:**  
In this lesson, we explored the basics of Python lists, including list creation, accessing elements, and determining the length of lists. Understanding how to work with lists is essential for effectively managing collections of data in Python. Lists offer flexibility and ease of use, making them a fundamental data structure in programming. As we progress, we will delve deeper into list methods and advanced manipulations to enhance our coding skills.

Extra python codes for Lesson 20

section 4 lesson 20

nums = [1,2,3,4,5,6,7]

print(nums)

names = ["ahmed","jermaine","mycle","mohamed"]

print(names)

l1 = ["sara",12,"ahmed",True,5.24]

print(l1)

fruits = ["banana","apple","mango","grapes"]

print(fruits)

fruits = ["banana","apple","mango","grapes"]

# git a value

print(fruits[1])

# the lenth of the fruits

print(len(fruits))

# constractors

l2 = list((12, 144, "yousef",False))

print(l2)

**Section 4 Lesson 21: Understanding List Manipulation in Python**

**Introduction:**  
In this lesson, we will explore various operations that can be performed on lists in Python. Lists are a fundamental data structure that allows us to store and manipulate collections of items. We will analyze the provided code line by line, explaining each operation and its purpose, along with examples to illustrate the concepts effectively.

**Code Breakdown:**

1. **Line 1: List Initialization**
   * **Code:** names = ["ahmed", "jermaine"]
   * **Explanation:** This line initializes a list named names containing two string elements.
   * **Purpose:** To create a collection of names for further manipulation.
   * **Example 1:** If we change the list to names = ["alice", "bob"], it now contains these two names.
   * **Example 2:** We could also initialize with names = [] to start with an empty list.
2. **Line 2: Print Statement**
   * **Code:** print(names)
   * **Explanation:** This line outputs the contents of the names list to the console.
   * **Purpose:** To display the current list of names.
   * **Example 1:** Running this line will output ["ahmed", "jermaine"].
   * **Example 2:** If the list were modified before this line, the output would reflect those changes.
3. **Line 3: Append Operation**
   * **Code:** names.append("hossam")
   * **Explanation:** This line adds the string "hossam" to the end of the names list.
   * **Purpose:** To extend the list by adding a new item.
   * **Example 1:** After this operation, the list becomes ["ahmed", "jermaine", "hossam"].
   * **Example 2:** If we append another name like names.append("mohamed"), the list will then be ["ahmed", "jermaine", "hossam", "mohamed"].
4. **Line 4: Remove Operation**
   * **Code:** names.remove("ahmed")
   * **Explanation:** This line removes the first occurrence of the string "ahmed" from the names list.
   * **Purpose:** To delete an item from the list by value.
   * **Example 1:** The list will now become ["jermaine", "hossam"].
   * **Example 2:** If we try to remove a name not in the list, such as names.remove("john"), it will raise a ValueError.
5. **Line 5: Insert Operation**
   * **Code:** names.insert(2, "nasser")
   * **Explanation:** This line inserts the string "nasser" at index 2 of the names list.
   * **Purpose:** To add an item at a specific position within the list.
   * **Example 1:** After this operation, the list becomes ["jermaine", "hossam", "nasser"].
   * **Example 2:** If we insert another "nasser" at index 1, it will change the list to ["jermaine", "nasser", "hossam", "nasser"].
6. **Line 6: Insert Another Value**
   * **Code:** names.insert(1, "nasser")
   * **Explanation:** This line again inserts "nasser" at index 1 of the names list.
   * **Purpose:** To demonstrate that we can have multiple identical items in a list.
   * **Example 1:** The list will now be ["jermaine", "nasser", "hossam", "nasser"].
   * **Example 2:** Inserting at index 0 with names.insert(0, "nasser") will change it to ["nasser", "jermaine", "nasser", "hossam", "nasser"].
7. **Line 7: Change Value**
   * **Code:** names[2] = "abdo"
   * **Explanation:** This line changes the value at index 2 of the names list to "abdo".
   * **Purpose:** To update an existing item in the list.
   * **Example 1:** The list now becomes ["jermaine", "nasser", "abdo", "nasser"].
   * **Example 2:** Changing the value at index 3 with names[3] = "abdo" will further modify the list to ["jermaine", "nasser", "abdo", "abdo"].
8. **Line 8: Print Updated List**
   * **Code:** print(names)
   * **Explanation:** This line outputs the updated contents of the names list.
   * **Purpose:** To show the result of the previous modifications.
   * **Example 1:** The output will be ["jermaine", "nasser", "abdo", "abdo"].
   * **Example 2:** If we had made additional changes, the output would reflect those.
9. **Line 9: Pop Operation**
   * **Code:** names.pop(2)
   * **Explanation:** This line removes the item at index 2 from the names list and returns it.
   * **Purpose:** To delete an item by its index and retrieve it.
   * **Example 1:** The list becomes ["jermaine", "nasser", "abdo"] after removing "abdo".
   * **Example 2:** If we pop from index 0 with names.pop(0), the resulting list will be ["nasser", "abdo"].
10. **Line 10: Reverse Operation**
    * **Code:** names.reverse()
    * **Explanation:** This line reverses the order of the items in the names list.
    * **Purpose:** To change the sequence of elements.
    * **Example 1:** After this operation, the list will be ["abdo", "nasser"].
    * **Example 2:** If we had more items, like ["a", "b", "c"], reversing would result in ["c", "b", "a"].
11. **Line 11: List Initialization for Numbers**
    * **Code:** nums = [5, 3, 4, 2, 1, 6]
    * **Explanation:** This line initializes a list named nums containing integers.
    * **Purpose:** To create a separate list for numerical operations.
    * **Example 1:** If we initialize with nums = [10, 8, 6], it will contain these numbers.
    * **Example 2:** An empty list can also be initialized with nums = [].
12. **Line 12: Print Statement for Numbers**
    * **Code:** print(nums)
    * **Explanation:** This line prints the contents of the nums list to the console.
    * **Purpose:** To display the current list of numbers.
    * **Example 1:** The output will be [5, 3, 4, 2, 1, 6].
    * **Example 2:** If modified, the output will reflect those changes.
13. **Line 13: Sort Operation**
    * **Code:** nums.sort()
    * **Explanation:** This line sorts the nums list in ascending order.
    * **Purpose:** To organize the list elements numerically.
    * **Example 1:** After sorting, the list becomes [1, 2, 3, 4, 5, 6].
    * **Example 2:** Sorting an already sorted list will keep it unchanged, e.g., nums = [1, 2, 3].
14. **Line 14: Print Sorted List**
    * **Code:** print(nums)
    * **Explanation:** This line prints the sorted contents of the nums list.
    * **Purpose:** To show the result of the sorting operation.
    * **Example 1:** The output will be [1, 2, 3, 4, 5, 6].
    * **Example 2:** If we sort another list like nums = [9, 7, 5], the output will be [5, 7, 9].
15. **Line 15: Reversing the Sorted List**
    * **Code:** nums.reverse(nums.sort)
    * **Explanation:** This line attempts to reverse the nums list but is incorrectly written; it should be nums.reverse().
    * **Purpose:** To change the order of the sorted list.
    * **Example 1:** A correct reversal after sorting will yield [6, 5, 4, 3, 2, 1].
    * **Example 2:** If we miss the parentheses, it will result in an error.
16. **Line 16: Sort and Reverse**
    * **Code:** nums.sort(reverse=True)
    * **Explanation:** This line sorts the nums list in descending order.
    * **Purpose:** To organize the list elements numerically in reverse order.
    * **Example 1:** After this operation, the list will become [6, 5, 4, 3, 2, 1].
    * **Example 2:** If we sort another list, nums = [3, 1, 2], the output will be [3, 2, 1].

**Conclusion:**  
In this lesson, we have explored various list manipulation techniques in Python, including appending, removing, inserting, changing values, and sorting lists. Understanding these operations is essential for effectively managing and manipulating collections of data. Lists provide us with a flexible way to store and organize information, making them a fundamental aspect of programming in Python. As we continue, we will delve deeper into more advanced list operations and functionalities.

Extra python for Lesson 21

Section 4 Lesson 21

names = ["ahmed", "jermaine"]

print(names)

# append

names.append("hossam")

# remove

names.remove("ahmed")

# insert

names.insert(2, "nasser")

names.insert(1, "nasser")

names.insert(0, "nasser")

# change value

names[2] = "abdo"

print(names)

names[3] = "abdo"

print(names)

# pop

names.pop(2)

print(names)

# reverse

names.reverse()

print(names)

nums = [5, 3, 4, 2, 1, 6]

print(nums)

# sort

nums.sort()

print(nums)

nums.reverse(nums.sort)

print(nums)

# sort and reverse

nums.sort(reverse=True)

print(nums)

section 4 lessin 22

tuples in python

**Understanding Lists vs. Tuples in Python**

In this explanation, we will break down the provided Python code line by line and clarify the differences between lists and tuples. We will also address whether or not you can change items in a tuple. Let's dive in!

1. **Line 1: Tuple Initialization**
   * **Code:** noms = (1, 2, 3, 4, 5)
   * **Explanation:** This line initializes a tuple named noms containing five integer elements.
   * **Purpose:** Tuples are used to store a collection of items, similar to lists but with key differences.
   * **Example 1:** If we change it to noms = (10, 20, 30), the tuple would have those three numbers.
   * **Example 2:** You can also create an empty tuple with noms = ().
2. **Line 2: Print Type**
   * **Code:** print(type(noms))
   * **Explanation:** This line prints the type of the variable noms.
   * **Purpose:** To show that noms is a tuple.
   * **Example 1:** The output will be <class 'tuple'>, indicating it's a tuple.
   * **Example 2:** If we had a list like noms = [1, 2, 3], it would output <class 'list'>.
3. **Line 3: Print Tuple**
   * **Code:** print(noms)
   * **Explanation:** This line prints the contents of the noms tuple.
   * **Purpose:** To display the items stored in the tuple.
   * **Example 1:** The output will be (1, 2, 3, 4, 5).
   * **Example 2:** If we had noms = (10,), the output would be (10,).
4. **Line 4: Comment on Trailing Comma**
   * **Code:** # trail in comma
   * **Explanation:** This comment indicates that a trailing comma is sometimes necessary in tuple initialization, especially for single-element tuples.
5. **Line 5: Single Element Tuple Constructor**
   * **Code:** t1 = ("ahmed",)
   * **Explanation:** This line creates a single-element tuple named t1 containing the string "ahmed". The comma is essential to differentiate it from a regular string.
   * **Purpose:** To show how to create a tuple with just one item.
   * **Example 1:** If we write t1 = ("ahmed"), it will be treated as a string, not a tuple.
   * **Example 2:** A similar tuple can be created with t1 = tuple(["ahmed"]).
6. **Line 6: Comment for Constructor**
   * **Code:** # # constructor
   * **Explanation:** This comment suggests that the following line constructs a tuple.
7. **Line 7: Tuple Creation from String**
   * **Code:** # # trail in comma
   * **Explanation:** This comment indicates that a tuple can be created from a string, where each character becomes an element.
8. **Line 8: Tuple Constructor from String**
   * **Code:** # t1 = tuple("ahmed", )
   * **Explanation:** This line (commented out) shows that you can convert a string into a tuple. Each character in "ahmed" would be an element in the tuple.
   * **Purpose:** To demonstrate versatility in tuple creation.
   * **Example 1:** If uncommented, t1 would become ('a', 'h', 'm', 'e', 'd').
9. **Line 9: Print Single Element Tuple**
   * **Code:** print(t1)
   * **Explanation:** This line outputs the contents of the t1 tuple.
   * **Purpose:** To show the single element in the tuple.
   * **Example 1:** The output will be ("ahmed",).
10. **Line 10: New Tuple Initialization**
    * **Code:** names = ("ahmed", "jermaine", "hossam")
    * **Explanation:** This line initializes a new tuple named names containing three strings.
    * **Purpose:** To create a collection of names.
    * **Example 1:** If we change it to names = ("john", "doe"), the tuple will have those names.
    * **Example 2:** An empty tuple can also be created with names = ().
11. **Line 11: Print Names Tuple**
    * **Code:** print(names)
    * **Explanation:** This line prints the contents of the names tuple.
    * **Purpose:** To display the names stored in the tuple.
    * **Example 1:** The output will be ("ahmed", "jermaine", "hossam").
    * **Example 2:** If we had names = ("alice",), it would output ("alice",).
12. **Line 12: Print Type of Names**
    * **Code:** print(type(names))
    * **Explanation:** This line prints the type of the names variable.
    * **Purpose:** To confirm that names is a tuple.
    * **Example 1:** The output will be <class 'tuple'>.
    * **Example 2:** If it were a list like names = ["ahmed", "jermaine"], it would output <class 'list'>.
13. **Line 13: Accessing an Item**
    * **Code:** print(names[1])
    * **Explanation:** This line retrieves the item at index 1 from the names tuple.
    * **Purpose:** To demonstrate accessing elements in a tuple.
    * **Example 1:** The output will be "jermaine".
    * **Example 2:** If we accessed print(names[0]), it would give "ahmed".
14. **Line 14: Length of Names Tuple**
    * **Code:** print(len(names))
    * **Explanation:** This line prints the number of elements in the names tuple.
    * **Purpose:** To show how to measure the size of a tuple.
    * **Example 1:** The output will be 3 since there are three names.
    * **Example 2:** If we added another name, like names = ("ahmed", "jermaine", "hossam", "mohamed"), the length would be 4.
15. **Line 15: Comment on Changing Items**
    * **Code:** # change items
    * **Explanation:** This comment indicates a misunderstanding, as tuples are immutable, meaning their items cannot be changed.
16. **Line 16: List Initialization**
    * **Code:** names = ["hoda"]
    * **Explanation:** This line initializes a list named names containing a single string element "hoda".
    * **Purpose:** To demonstrate the difference between lists (mutable) and tuples (immutable).
    * **Example 1:** If we write names = ["alice", "bob"], it will create a list with two names.
    * **Example 2:** An empty list can also be created with names = [].
17. **Line 17: Print List**
    * **Code:** print(names)
    * **Explanation:** This line prints the contents of the names list.
    * **Purpose:** To display the item stored in the list.
    * **Example 1:** The output will be ["hoda"].
    * **Example 2:** If we had names = ["john"], it would output ["john"].
18. **Line 18: Delete List**
    * **Code:** del names
    * **Explanation:** This line deletes the variable names, removing the reference to the list.
    * **Purpose:** To demonstrate deletion of a variable.
    * **Example 1:** After this line, names no longer exists.
    * **Example 2:** Trying to print print(names) after this line would raise a NameError.
19. **Line 19: Print Deleted Names**
    * **Code:** print(names)
    * **Explanation:** This line attempts to print names, which has already been deleted.
    * **Purpose:** To demonstrate the result of trying to access a deleted variable.
    * **Example 1:** This line will raise a NameError: name 'names' is not defined.
    * **Example 2:** If we try to print a variable that still exists, it will print its contents.

**Conclusion:**  
To summarize, tuples are immutable collections, meaning their contents cannot be changed after creation, while lists are mutable and allow modifications. You can access items in both, but if you try to change a tuple, you will encounter an error. Understanding these differences is crucial for effective programming in Python.

**Lecture by Professor Mahmoud: Understanding Lists and Tuples in Python**

**Good morning, students!**

Today, we will delve into a fundamental concept in Python programming: the difference between lists and tuples. We will explore this through a sample code snippet. As we go through each line, I encourage you to ask questions for clarity. Let us begin.

**1. Tuple Initialization**

noms = (1, 2, 3, 4, 5)

In this line, we are initializing a tuple named noms that contains five integer elements. **Tuples** are immutable collections, meaning that once they are created, their contents cannot be altered. This property makes tuples useful for storing fixed collections of items.

**2. Print the Type of the Tuple**

print(type(noms))

Here, we are printing the type of the variable noms. The output will confirm that noms is indeed a tuple. Understanding the type of data we are working with is crucial for writing effective code.

**3. Displaying the Tuple Contents**

print(noms)

This line outputs the contents of the noms tuple. It will display the integers we have stored in it. By visualizing the data structure, we can better understand how to work with it in our programs.

**4. Single-Element Tuple Initialization**

t1 = ("ahmed",)

In this line, we create a single-element tuple called t1. Note the trailing comma; it is essential for Python to recognize this as a tuple rather than a string. This illustrates how to define a tuple with a single item effectively.

**5. Tuple from a String**

# t1 = tuple("ahmed", )

This commented line demonstrates how we can convert a string into a tuple, where each character of the string would become an individual element in the tuple. This showcases the flexibility of tuple creation.

**6. Displaying the Single-Element Tuple**

print(t1)

Here, we print the contents of the t1 tuple. The output will display ("ahmed",), revealing the single string we stored in the tuple.

**7. New Tuple Initialization with Names**

names = ("ahmed", "jermaine", "hossam")

In this line, we initialize a new tuple named names that contains three strings. Tuples can hold any type of data, making them versatile for various applications.

**8. Print the Names Tuple**

print(names)

We now print the contents of the names tuple. The output will display the three names we have stored, reinforcing our understanding of tuple contents.

**9. Type of Names Tuple**

print(type(names))

This line will show the type of the names variable, confirming that it is a tuple. Understanding data types is fundamental in programming as it influences how we manipulate data.

**10. Accessing an Item in the Tuple**

print(names[1])

Here, we are accessing the element at index 1 of the names tuple, which will output "jermaine". This demonstrates how to retrieve specific items from a tuple.

**11. Length of the Tuple**

print(len(names))

This line outputs the number of elements in the names tuple, which in this case will be 3. Knowing the length of a data structure can help us manage our data effectively.

**12. Transition to a List**

names = ["hoda"]

In this line, we transition from using a tuple to a list by initializing a list named names that contains a single string element "hoda". Lists are mutable, meaning their contents can be altered after creation, offering greater flexibility compared to tuples.

**13. Print the List**

print(names)

This line will print the contents of the names list, displaying ["hoda"]. This reinforces the idea that lists can be modified.

**14. Deleting the List**

del names

Here, we delete the names list, effectively removing the reference to it. Understanding how to manage memory and variable scope is vital for efficient programming.

**15. Attempting to Print Deleted Names**

print(names)

Finally, this line attempts to print names, which has been deleted. As a result, you will encounter a NameError indicating that the variable is not defined. This serves as a reminder of the importance of variable management in programming.

**Conclusion:**

In summary, we have explored the distinctions between lists and tuples in Python. **Tuples** are immutable, making them suitable for fixed collections of data, while **lists** are mutable, allowing for dynamic changes. Understanding these concepts is essential for effective programming practices.

Thank you for your attention, and I welcome any questions you may have!

**Lecture Title: Section 4, Lesson 23 - Understanding Sets in Python**

**Good morning, students!**

Today, we will explore the concept of sets in Python through a simple code example. I will explain each line of the code, and we will see how sets work. Let's dive in!

**1. Defining a Set**

fruits = {"banana", "apple", "mango"}

In this line, we are creating a set named fruits that contains three items: "banana", "apple", and "mango". Sets are a collection of unique elements, which means that no two elements in a set can be the same.

**2. Checking the Type of the Set**

print(type(fruits))

Here, we are printing the type of the variable fruits. The output will show that it is of type set. This helps us confirm that we are indeed working with a set data structure.

**3. Printing the Set**

print(fruits)

This line prints the contents of the set fruits. The output will display the items in the set. Remember that the order may not be the same each time since sets are unordered collections.

**4. Creating a New Set**

noms\_set = {1, 2, 3, 4, 5, 6, 7}

In this line, we are creating a new set named noms\_set that contains the numbers from 1 to 7. Just like the previous set, this set can only contain unique elements.

**5. Printing the New Set**

print(noms\_set)

Here, we print the contents of the noms\_set. This will display the numbers we defined in the set.

**6. Checking the Type of the New Set**

print(type(noms\_set))

Once again, we check the type of the variable noms\_set. The output will confirm that it is also of type set.

**7. Checking for an Element in the Set**

print("grapes" in fruits)

In this line, we are checking if the item "grapes" is present in the fruits set. The output will be either True or False. At this point, it will return False because "grapes" is not yet in the set.

**8. Adding an Element to the Set**

fruits.add("grapes")

Now, we are adding "grapes" to the fruits set. This is done using the add method.

**9. Printing the Updated Set**

print(fruits)

Here, we print the fruits set again to see the updated contents. The output will now include "grapes".

**10. Removing an Element from the Set**

fruits.remove("banana")

In this line, we are removing "banana" from the fruits set using the remove method. If "banana" is in the set, it will be removed successfully.

**11. Printing the Set After Removal**

print(fruits)

We print the fruits set again to confirm that "banana" has been removed.

**12. Adding an Element Again**

fruits.add("apple")

Here, we attempt to add "apple" to the fruits set once more. However, since "apple" is already in the set, there will be no change in the set.

**13. Printing the Set Again**

print(fruits)

We print the fruits set again. Despite trying to add "apple" again, it will remain unchanged since sets only store unique elements.

**14. Deleting the Set**

del fruits

In this line, we are deleting the entire fruits set using the del statement. After this, we will no longer have access to the fruits variable.

**15. Attempting to Print the Deleted Set**

print(fruits)

Here, we try to print the fruits set again. This will result in an error because the set has been deleted, and it no longer exists in our program.

**16. Clearing the Set (Commented Out)**

# fruits.clear()

This line is commented out, but if it were executed, it would clear all elements from the fruits set without deleting the set itself.

**17. Printing the Set After Clearing (Commented Out)**

# print(fruits)

Again, this line is commented out. If it were executed after clearing the set, it would show an empty set.

**Conclusion:**

In conclusion, we have explored sets in Python, including how to create, add, remove, and check for elements within a set. Remember that sets are unique collections and do not allow duplicate values.

Thank you for your attention, and I'm happy to answer any questions you may have!

Extra python code for lesson 23

Section 4

Lesson 23

type of this list is set

fruits = {"banana", "apple", "mango"}

print(type(fruits))

print(fruits)

# new set

noms\_set = {1, 2, 3, 4, 5, 6, 7}

print(noms\_set)

print(type(noms\_set))

# check in set

print("grapes"in fruits)

# add to set

fruits.add("grapes")

print(fruits)

# remove from set

fruits.remove("banana")

print(fruits)# fruits.add("apple")

fruits.add("apple")

print(fruits)

# dleate set

del fruits

print(fruits)

# clear set

fruits.clear()

print(fruits)

**Lecture Title: Section 4, Lesson 24 - Understanding Collections in Python**

**Good morning, everyone!**

Today, we will be discussing three important data structures in Python: lists, tuples, and sets. Understanding these collections will help us manage and organize data effectively in our programs. Let's explore each collection type in detail, along with examples to reinforce our understanding.

**1. Lists**

numbers = [1, 2, 3, 4, 5]

**Definition:** A list is a collection that is **ordered** and **changeable**. It allows duplicate members.

In this example, we have a list called numbers that contains the integers from 1 to 5. Since lists are ordered, the elements maintain their position, and we can access them by their index. For instance, numbers[0] will give us 1, while numbers[2] will give us 3.

**Example:**

print(numbers[0]) # Output: 1

**2. Tuples**

fruits = ('Apples', 'Oranges', 'Grapes')

**Definition:** A tuple is a collection that is **ordered** but **unchangeable**. It also allows duplicate members.

In this example, we have a tuple named fruits which contains three fruit names. Unlike lists, once we create a tuple, we cannot modify its content. This makes tuples useful for storing data that should not change.

**Example:**

print(fruits[1]) # Output: Oranges

**3. Sets**

names\_set = {"ahmed", "hoda", "samy"}

**Definition:** A set is a collection that is **unordered** and **unindexed**. It does not allow duplicate members.

In this example, we have a set called names\_set that contains three names. Since sets are unordered, the items do not have a specific position, and we cannot access them by an index. If we try to add a duplicate name, it will not be included in the set.

**Example:**

names\_set.add("ahmed") # Adding a duplicate name will not change the set

print(names\_set) # Output: {'ahmed', 'hoda', 'samy'}

**4. Additional Examples for Better Understanding**

**Lists:**

numbers = [1, 2, 3, 4, 5, 6, 7, 8]

names = ["ahmed", "mohamed", "samy"]

fruits = ["Banana", "Mango", "Apple"]

Here, we have additional lists for numbers, names, and fruits. You can see that each list can contain different types of data, and they can be modified at any time.

**Tuples:**

numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9)

name = ("mohamed", "ahmed", "ali")

In this case, we have tuples for numbers and names. Remember, you cannot change the values in these tuples once they are defined.

**Sets:**

numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

names = {"mahmoud", "ibrahim"}

Finally, we have sets for numbers and names. Again, remember that sets do not allow duplicates and are unordered.

**Conclusion:**

In conclusion, lists, tuples, and sets are essential data structures in Python, each serving different purposes and offering unique features.

* **Lists** are ordered and changeable, allowing duplicates.
* **Tuples** are ordered and unchangeable, also allowing duplicates.
* **Sets** are unordered and unchangeable, with no duplicates allowed.

Thank you for your attention! If you have any questions or need further clarification, feel free to ask!

Extra python code for lesson 24

# A List is a collection

#which is ordered

#and changeable. Allows duplicate members

numbers = [1, 2, 3, 4, 5]

#A Tuple is a collection

#which is ordered

#and unchangeable. Allows duplicate members

fruits = ('Apples', 'Oranges', 'Grapes')

#A Set is a collection

#which is unordered and unindexed.

#No duplicate members.

names\_set = {"ahmed" "hoda", "samy"}

#extra code for clear understading to lesson 24

##list

numbers = [1,2,3,4,5,6,7,8,]

names = ["ahmed","mohamed","samy")

fruits = ["Banana", "mango", "apple"]

#Tuple

numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9)

name = ("mohamed", "ahmed", "ali")

#set

numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Names = {"mahmoud", "ibrahim"}