Ahmed Sami, explaining each Python code line by line with examples:

**Lecture by Ahmed Sami: Understanding Basic Python Programming Concepts**

Good morning, students!

Today, we will delve into some fundamental Python programming concepts. I will explain each line of code with examples to ensure you grasp the practical applications of these concepts. Let’s get started!

**Introduction to Python Programming**

In our course, we use an online environment called Replit to write and execute Python code. Let’s begin with a simple program to output text to the console.

print("besem allah")

This code uses the print function to display the string “besem allah” on the screen. The print function is crucial in Python as it allows us to provide feedback and information to users through the console.

**Lesson on the Print Function**

Let’s explore the print function further with a sequence of commands:

print("start")

print("working")

print("end")

When executed, this code outputs:

start

working

end

Notice how the output appears in the order the commands are written, from top to bottom. If we change the sequence:

print("end")

print("working")

print("start")

The output will be:

end

working

start

This demonstrates how the order of print statements directly influences the displayed output, reinforcing the concept of sequence in programming.

Now, let’s add another line:

print("start")

print("working")

print("ahmed")

print("end")

The output will be:

start

working

ahmed

end

This shows how we can dynamically modify outputs within a program.

**Creating a User Profile Program**

Next, let’s create a simple user profile program using the print function:

print("my name is Mahmoud Shaabo")

print("my age is 39 years old")

print("I am a python programmer")

print("my website is [link]")

print("my youtube channel is [link]")

When run, the output presents user data sequentially:

my name is Mahmoud Shaabo

my age is 39 years old

I am a python programmer

my website is [link]

my youtube channel is [link]

This exercise demonstrates how multiple print statements can convey a series of related information about a user.

**Introduction to the Input Function**

In Lesson 6, we introduced the input function, which allows the program to receive data from the user:

name = input("please enter your name: ")

print("hi " + name)

Here, the program prompts the user to enter their name. Once the user provides an input (e.g., “Mahmoud Shaabo”), the program responds with:

hi Mahmoud Shaabo

This demonstrates basic interactivity within a program.

Let’s expand this idea by adding the user’s age:

name = input("please enter your name: ")

print("hi " + name)

age = input("your age is: ")

print("and your age is " + age)

If the user enters “39” for their age, the output will be:

please enter your name: Mahmoud Shaabo

hi Mahmoud Shaabo

your age is: 39

and your age is 39

**Final Comprehensive Example**

To consolidate the concepts learned, let’s combine the print and input functions:

print("besem allah")

print("start")

print("ahmed")

print("working")

print("sara")

print("end")

name = input("please enter your name: ")

age = input("your age is: ")

work = input("your work is: ")

print("hi " + name)

print("and your age is " + age)

print("and your work is " + work)

This code integrates various elements, prompting the user for their name, age, and occupation while also displaying initial messages. The expected output will display the information sequentially, demonstrating the program’s full capability to interact with a user.

**Conclusion**

The lessons outlined today provide a solid foundation for understanding Python programming, focusing on the print and input functions. These building blocks are essential for creating interactive applications and will serve as the basis for more advanced programming concepts in future lessons.

Thank you for your attention, and I look forward to seeing your progress in Python programming!

Feel free to ask any questions or request further clarifications on any of the topics covered today. Happy coding!

**Lecture by Ahmed Sami: Comprehensive Lesson on Variables in Python Programming Language**

Good morning, students!

Today, we will explore the concept of variables in Python. Variables are essential for storing and manipulating data in your programs. Let’s dive into the details.

**Importance of Variables**

Variables allow us to store values that we can use and manipulate throughout our programs. They can hold different types of data, such as numbers, strings, and more.

**Definition of Variables**

In Python, variables are like containers that hold values. To define a variable, you write the variable name, followed by an equals sign, and then the value you want to assign to it. For example:

name = "ahmed"

Here, the variable name is assigned the value "ahmed", which is a string.

**Data Types in Python**

Python supports various data types, including:

* **Strings**: Sequences of characters enclosed in quotation marks. Used to store text values.
* **Integers**: Whole numbers without decimal points. Used to represent numeric values.
* **Floats**: Decimal numbers that include a decimal point. Used to represent fractional values.
* **Booleans**: Can have two possible values: True or False. Used to represent logical values.

**Naming Conventions for Variables**

When naming variables, follow these rules and conventions:

* Variables should not start with a capital letter or contain spaces. Use underscores (\_) to separate words, or use camelCase.
* Avoid using reserved keywords as variable names (e.g., int).
* Choose meaningful and descriptive names to enhance code readability.

Examples of valid variable names:

num = 13 # integer

name = "ahmed" # string

is\_cool = True # boolean

fl = 3.5 # float

**Retrieving the Type of a Variable**

To determine the data type of a variable, use the type() function:

name = "ahmed"

print(type(name))

The output will be <class 'str'>, indicating that name is of type string.

**Example of Commenting Code**

In Python, you can add comments to your code to provide explanations or make notes. Comments are lines of code that the interpreter ignores. To add a comment, prefix the line with the # symbol:

# This is a comment explaining the purpose of the following code

num = 13

print(type(num)) # This line prints the type of the variable "num"

**Variable Declaration Methods**

Let’s explore various methods for defining variables in Python.

**Assigning Multiple Variables on a Single Line**

You can declare multiple variables on a single line using semicolons:

num = 13; x = 20; name = "Mohamed"; is\_running = True

This line declares four variables: num with the value 13, x with the value 20, name with the string “Mohamed”, and is\_running as a boolean True.

To output the values of these variables, use the print function:

print(num, name, is\_running)

The output will be:

13 Mohamed True

**Tuple Unpacking**

Another method for variable declaration involves assigning multiple variables in a single line using tuple unpacking:

x, y, z = (17, 20.5, True)

This assigns 17 to x, 20.5 to y, and True to z. When printed:

print(x, y, z)

The output is:

17 20.5 True

**Arithmetic Operations**

Let’s learn about performing arithmetic operations in Python.

**Adding Variables**

Create three variables and perform addition:

x = 10

y = 20

z = x + y

print(z)

The output will be 30.

**Other Arithmetic Operations**

You can also perform subtraction, multiplication, and division:

x = 10

y = 20

print(x + y) # Addition

print(x - y) # Subtraction

print(x \* y) # Multiplication

print(x / y) # Division

The results are:

30

-10

200

0.5

**Working with Decimal Numbers**

You can work with decimal numbers in Python:

nom1 = 10.6

nom2 = 30.26

print(nom1 + nom2)

The output will be 40.86.

**Combining Text Values**

You can also perform operations on text values:

first\_name = "Ahmed"

last\_name = "Sami"

print(first\_name + last\_name)

The output will be “AhmedSami”. To add a space between the names:

full\_name = first\_name + " " + last\_name

print(full\_name)

The result is “Ahmed Sami”.

**Conclusion**

In this lesson, we learned how to perform arithmetic operations and combine text values in Python. Practice these examples to enhance your programming skills.

Feel free to ask any questions or request further clarifications on any of the topics covered today. Happy coding!

**Lecture by Ahmed Sami: Python Programming - Casting Variables and Creating a Simple Calculator**

Good morning, students!

Today, we will explore the concept of casting variables in Python and how to create a simple calculator. Let’s dive into the details.

**Casting Variables**

Casting refers to converting a variable from one data type to another. This is particularly useful when you need to perform operations that require specific data types.

**String Input**

Let’s start with getting user input as a string:

name = input("Enter your name: ")

print("Hi " + name)

Here, the input() function always returns a string. When we concatenate ‘Hi’ with name, we get a greeting. However, if we want a space between ‘Hi’ and the name, we add a space after ‘Hi’ in the print() function:

print("Hi " + name)

**Integer Input**

Now, let’s consider we want to work with numbers:

num1 = input("Enter number 1: ")

num2 = input("Enter number 2: ")

print(num1 + num2)

If you enter ‘12’ for both, the output will be ‘1212’, not 24. This is because input() returns a string, and the + operator concatenates strings.

**Applying Casting**

To fix this, we cast the inputs to integers:

num1 = int(input("Enter number 1: "))

num2 = int(input("Enter number 2: "))

print(num1 + num2)

Now, if you enter ‘12’ for both, the output will be 24. The int() function converts the string to an integer, allowing for arithmetic addition.

**Building a Simple Calculator**

Finally, let’s build a simple calculator that performs basic arithmetic operations:

num1 = int(input("Enter number 1: "))

num2 = int(input("Enter number 2: "))

print("Addition:", num1 + num2)

print("Subtraction:", num1 - num2)

print("Multiplication:", num1 \* num2)

print("Division:", num1 / num2)

This calculator takes two numbers from the user and prints the results of addition, subtraction, multiplication, and division.

**Conclusion**

Casting is essential when you want to perform operations that require numerical values, especially when those values come from user input. Practice these examples to enhance your programming skills.

**Lecture by Ahmed Sami: Understanding Python Comments and Basic Operations**

Good morning, students!

Today, we will review the importance of comments in programming and basic operations in Python. Let’s get started.

**Code Overview**

Here is the code that was created in the previous lesson:

n1 = float(input("first number: "))

n2 = float(input("second number: "))

print(n1 + n2)

print(n1 - n2)

print(n1 \* n2)

print(n1 / n2)

**Code Explanation**

**Input the First Number**

n1 = float(input("first number: "))

This line prompts the user to enter the first number. The input() function reads the user’s input as a string, and then we convert it to a float using float(). This allows for decimal numbers, making our calculations more precise.

**Input the Second Number**

n2 = float(input("second number: "))

Similarly, this line requests the user to input the second number and converts it to a float.

**Print the Sum**

print(n1 + n2)

This line calculates the sum of n1 and n2 and prints the result to the console.

**Print the Difference**

print(n1 - n2)

This line computes the difference between n1 and n2 and prints it.

**Print the Product**

print(n1 \* n2)

Here, we multiply n1 and n2, displaying the product.

**Print the Quotient**

print(n1 / n2)

This line divides n1 by n2 and prints the result. Be cautious with division by zero, as this will raise an error.

**The Importance of Comments**

Comments are notes in the code that are not executed by the program. They help explain what certain parts of the code do.

**Why Write Comments?**

* Comments help you remember what each line of code does, especially when you revisit the code later.
* They make your code more understandable for others who might read it in the future.

**Adding Comments to Code**

Before each variable definition, you can add comments like this:

# input the first number

n1 = float(input("first number: "))

# input the second number

n2 = float(input("second number: "))

The # symbol indicates that everything following it on that line is a comment.

**Multi-line Comments**

You can add multi-line comments using triple quotes:

"""

This app is made by Ahmed Sami.

It's a simple calculator.

"""

This block can explain the purpose of the program and can span multiple lines.

**Conclusion**

Writing comments in your Python code not only makes you a better programmer but also helps others understand your work. By using comments effectively, you can clarify what each section of your code does, allowing for easier collaboration and debugging.

**Lecture by Ahmed Sami: Understanding Basic Python Code**

Good morning, students!

Today, we will break down a short piece of code line by line, explaining each part and its purpose. This code demonstrates how to work with strings in Python.

**Code Breakdown**

first\_name = "ahmed"

last\_name = 'sami'

# print(type(last\_name))

print(first\_name + " " + last\_name)

**Line 1: Variable Assignment**

first\_name = "ahmed"

This line declares a variable named first\_name and assigns it the string value “ahmed”. Variables are used to store data that can be referenced later.

**Line 2: Variable Assignment**

last\_name = 'sami'

Similar to the first line, this line creates another variable named last\_name and assigns it the string value ‘sami’. Using single quotes or double quotes in Python for strings is interchangeable.

**Line 3: Comment**

# print(type(last\_name))

This line is a comment, which is ignored by the Python interpreter. It is used to provide explanations or notes within the code.

**Line 4: Print Statement**

print(first\_name + " " + last\_name)

This line outputs the combined value of first\_name and last\_name to the console, with a space in between. The print() function is used to display information to the user. Concatenation (+) is used to join strings together.

**Summary**

In this simple code snippet, we learned how to create and manipulate string variables in Python. We assigned values to variables, used comments for clarity, and printed the combined result to the console. Understanding these basics is essential for building more complex programs.

Feel free to ask any questions or request further clarifications on any of the topics covered today. Happy coding!

**Lecture by Ahmed Sami: Understanding String Methods in Python**

Good morning, students!

Today, we will explore various string methods available in Python that can be used to manipulate and analyze strings. Let’s break down the provided code line by line.

**Code Breakdown**

name = "ahmed"

This line initializes a variable named name and assigns it the string value “ahmed”. It stores the name of a person that could be used later in the program.

s = "hello world"

This line initializes a variable named s and assigns it the string value “hello world”. It sets up a sample string that will be used for demonstrating various string methods.

# print(s)

This line is commented out and will not execute. It seems to be for reference or to show how to print the string s.

**Length of the String**

print(len(s))

This line prints the length of the string s using the len() function. It shows how to determine the number of characters in the string, which in this case is 11.

**Capitalize the String**

print(s.capitalize())

This line prints the string s with the first letter capitalized using the capitalize() method. It demonstrates how to format the string so that the first character is uppercase, resulting in “Hello world”.

**Convert to Uppercase**

print(s.upper())

This line prints the string s in all uppercase using the upper() method. It demonstrates how to convert the entire string to uppercase, resulting in “HELLO WORLD”.

**Convert to Lowercase**

print(s.lower())

This line prints the string s in all lowercase using the lower() method. It demonstrates how to convert the entire string to lowercase, resulting in “hello world”.

**Swap Case**

print(s.swapcase())

This line prints the string s with all uppercase letters converted to lowercase and vice versa using the swapcase() method. It demonstrates how to change the case of each character in the string, resulting in “HELLO WORLD”.

**Replace Substring**

print(s.replace("world", "everyone"))

This line prints the string s with the substring “world” replaced by “everyone” using the replace() method. It demonstrates how to modify parts of the string, resulting in “hello everyone”.

**Count Occurrences of a Character**

char = "l"

print(s.count(char))

This line initializes a variable named char and assigns it the character “l”. It then prints the number of times the character stored in char appears in the string s using the count() method. It demonstrates how to count the occurrences of a specific character, which is 3 in this case.

**Check if String Starts with a Substring**

print(s.startswith("he"))

This line prints a boolean value indicating whether the string s starts with the substring “he” using the startswith() method. It demonstrates how to check the beginning of a string, which returns True.

**Check if String Ends with a Substring**

print(s.endswith("s"))

This line prints a boolean value indicating whether the string s ends with the substring “s” using the endswith() method. It demonstrates how to check the end of a string, which returns False.

**Split the String**

print(s.split())

This line prints a list of substrings obtained by splitting the string s using the split() method. It demonstrates how to separate a string into its individual words, resulting in ["hello", "world"].

**Find Position of a Substring**

print(s.find("d"))

This line prints the index of the first occurrence of the substring “d” in the string s using the find() method. It demonstrates how to locate a specific character within a string, returning the index 9.

**Check if String is Alphanumeric**

print(s.isalnum())

This line prints a boolean value indicating whether the string s consists only of alphanumeric characters using the isalnum() method. It demonstrates how to check if a string contains only letters and numbers, which returns False in this case due to the space.

**Check if String is Alphabetic**

print(s.isalpha())

This line prints a boolean value indicating whether the string s consists only of alphabetic characters using the isalpha() method. It demonstrates how to check if a string contains only letters, which returns False in this case due to the space.

**Check if String is Numeric**

print(s.isnumeric())

This line prints a boolean value indicating whether the string s consists only of numeric characters using the isnumeric() method. It demonstrates how to check if a string contains only numbers, which returns False because the string contains letters.

**Summary**

The provided code demonstrates various methods available in Python for manipulating and analyzing strings. It showcases how to measure string length, change case, replace substrings, count characters, and check for specific conditions like whether a string starts or ends with certain substrings.

**Lecture by Ahmed Sami: Understanding Python Lists**

Good morning, students!

Today, 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. Let’s break down the code line by line.

**Code Breakdown**

**List Initialization**

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

This line initializes a list named nums that contains integers from 1 to 7. Lists are used to store multiple values in a single variable, making data management easier.

**Print Statement**

print(nums)

This line outputs the contents of the nums list to the console. The print() function is used to display information to the user, allowing us to see the contents of our list.

**List of Strings**

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

This line creates a list named names that contains four string elements. Lists can hold items of different data types, such as strings, integers, and more. Here, we are storing names.

**Print Statement**

print(names)

This line prints the contents of the names list to the console. It allows us to see what names are currently stored in the list.

**Mixed Data Types in a List**

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

This line initializes a list named l1 containing mixed data types: strings, an integer, a boolean, and a float. Lists can store different types of data, allowing for flexibility in data management.

**Print Statement**

print(l1)

This line prints the contents of the l1 list to the console. It provides visibility of the mixed data types stored in the list.

**Fruits List Initialization**

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

This line creates a list named fruits containing the names of four different fruits. It demonstrates how to create a list specifically for storing fruit names.

**Print Statement**

print(fruits)

This line prints the contents of the fruits list to the console. It allows us to see the fruit names currently stored in the list.

**Accessing List Values**

print(fruits[1])

This line accesses the second element in the fruits list (index 1, since indexing starts at 0). It demonstrates how to retrieve specific items from a list using their index.

**Length of the List**

print(len(fruits))

This line prints the length of the fruits list, which is the number of items it contains. It shows how to determine the size of a list.

**List Constructor**

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

This line creates a list named l2 using the list() constructor with a tuple as an argument. It illustrates how to create a list from other iterable data types.

**Print Statement**

print(l2)

This line prints the contents of the l2 list to the console. It provides visibility of the items created in the l2 list.

**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.

**Lecture by Ahmed Sami: Understanding List Manipulation in Python**

Good morning, students!

Today, 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. Let’s analyze the provided code line by line.

**Code Breakdown**

**List Initialization**

names = ["ahmed", "jermaine"]

This line initializes a list named names containing two string elements. It creates a collection of names for further manipulation.

**Print Statement**

print(names)

This line outputs the contents of the names list to the console. It displays the current list of names.

**Append Operation**

names.append("hossam")

This line adds the string “hossam” to the end of the names list. It extends the list by adding a new item. After this operation, the list becomes ["ahmed", "jermaine", "hossam"].

**Remove Operation**

names.remove("ahmed")

This line removes the first occurrence of the string “ahmed” from the names list. It deletes an item from the list by value. The list will now become ["jermaine", "hossam"].

**Insert Operation**

names.insert(2, "nasser")

This line inserts the string “nasser” at index 2 of the names list. It adds an item at a specific position within the list. After this operation, the list becomes ["jermaine", "hossam", "nasser"].

**Insert Another Value**

names.insert(1, "nasser")

This line again inserts “nasser” at index 1 of the names list. It demonstrates that we can have multiple identical items in a list. The list will now be ["jermaine", "nasser", "hossam", "nasser"].

**Change Value**

names[2] = "abdo"

This line changes the value at index 2 of the names list to “abdo”. It updates an existing item in the list. The list now becomes ["jermaine", "nasser", "abdo", "nasser"].

**Print Updated List**

print(names)

This line outputs the updated contents of the names list. The output will be ["jermaine", "nasser", "abdo", "nasser"].

**Pop Operation**

names.pop(2)

This line removes the item at index 2 from the names list and returns it. It deletes an item by its index and retrieves it. The list becomes ["jermaine", "nasser", "nasser"] after removing “abdo”.

**Reverse Operation**

names.reverse()

This line reverses the order of the items in the names list. It changes the sequence of elements. After this operation, the list will be ["nasser", "nasser", "jermaine"].

**List Initialization for Numbers**

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

This line initializes a list named nums containing integers. It creates a separate list for numerical operations.

**Print Statement for Numbers**

print(nums)

This line prints the contents of the nums list to the console. The output will be [5, 3, 4, 2, 1, 6].

**Sort Operation**

nums.sort()

This line sorts the nums list in ascending order. It organizes the list elements numerically. After sorting, the list becomes [1, 2, 3, 4, 5, 6].

**Print Sorted List**

print(nums)

This line prints the sorted contents of the nums list. The output will be [1, 2, 3, 4, 5, 6].

**Reversing the Sorted List**

nums.reverse()

This line reverses the nums list. It changes the order of the sorted list. A correct reversal after sorting will yield [6, 5, 4, 3, 2, 1].

**Sort and Reverse**

nums.sort(reverse=True)

This line sorts the nums list in descending order. It organizes the list elements numerically in reverse order. After this operation, the list will become [6, 5, 4, 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.

**Lecture by Ahmed Sami: Understanding Lists vs. Tuples in Python**

Good morning, students!

Today, 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!

**Code Breakdown**

**Tuple Initialization**

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

This line initializes a tuple named noms containing five integer elements. Tuples are used to store a collection of items, similar to lists but with key differences.

**Print Type**

print(type(noms))

This line prints the type of the variable noms. The output will be <class 'tuple'>, indicating it’s a tuple.

**Print Tuple**

print(noms)

This line prints the contents of the noms tuple. The output will be (1, 2, 3, 4, 5).

**Comment on Trailing Comma**

# trail in comma

This comment indicates that a trailing comma is sometimes necessary in tuple initialization, especially for single-element tuples.

**Single Element Tuple Constructor**

t1 = ("ahmed",)

This line creates a single-element tuple named t1 containing the string “ahmed”. The comma is essential to differentiate it from a regular string.

**Print Single Element Tuple**

print(t1)

This line outputs the contents of the t1 tuple. The output will be ("ahmed",).

**New Tuple Initialization**

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

This line initializes a new tuple named names containing three strings. It creates a collection of names.

**Print Names Tuple**

print(names)

This line prints the contents of the names tuple. The output will be ("ahmed", "jermaine", "hossam").

**Print Type of Names**

print(type(names))

This line prints the type of the names variable. The output will be <class 'tuple'>.

**Accessing an Item**

print(names[1])

This line retrieves the item at index 1 from the names tuple. The output will be “jermaine”.

**Length of Names Tuple**

print(len(names))

This line prints the number of elements in the names tuple. The output will be 3 since there are three names.

**Comment on Changing Items**

# change items

This comment indicates a misunderstanding, as tuples are immutable, meaning their items cannot be changed.

**List Initialization**

names = ["hoda"]

This line initializes a list named names containing a single string element “hoda”. It demonstrates the difference between lists (mutable) and tuples (immutable).

**Print List**

print(names)

This line prints the contents of the names list. The output will be ["hoda"].

**Delete List**

del names

This line deletes the variable names, removing the reference to the list.

**Print Deleted Names**

print(names)

This line attempts to print names, which has already been deleted. This will raise a NameError: name 'names' is not defined.

**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 Ahmed Sami: 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!

**Code Breakdown**

**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.

**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

**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.

**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.

**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.

**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.

**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.

**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.

**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.

**Printing the Updated Set**

print(fruits)

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

**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.

**Printing the Set After Removal**

print(fruits)

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

**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.

**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.

**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.

**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.

**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.

**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.

**Lecture by Ahmed Sami: 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.

**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

**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

**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'}

**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.