Title: Section 6, Lesson 30: Understanding Conditional Statements in Python

Welcome, students, to our lecture on conditional statements in Python. Today, we'll be exploring the use of 'if', 'elif', and 'else' statements through a series of examples. Let's break down the code and understand each concept step by step.

1. Basic 'if' Statement:

x = 30

y = 21

if x > y:

print("true")

Here, we're using a simple 'if' statement. It checks if x is greater than y. Since 30 is indeed greater than 21, the condition is true, and "true" will be printed.

1. 'if' Statement with Literal Values:

if 5 < 10:

print("true")

This example demonstrates that we can use literal values in our conditions. Since 5 is less than 10, "true" will be printed.

1. 'if' Statement with False Condition:

if 5 < 3:

print("true")

In this case, 5 is not less than 3, so the condition is false. As a result, nothing will be printed. The code inside the 'if' block is skipped.

1. Introducing 'else':

x = 30

y = 21

if x < y:

print("true")

else:

print("false")

Here, we introduce the 'else' statement. If the condition (x < y) is false, which it is in this case, the code in the 'else' block is executed. So, "false" will be printed.

1. 'if-else' with Strings:

name = "ahmed"

if name == "ahmed":

print("true")

else:

print("false")

This example shows that we can use string comparisons in our conditions. Since the name is "ahmed", "true" will be printed.

1. 'if-else' with Different String:

name = "jermain"

if name == "ahmed":

print("true")

else:

print("false")

Now, the name is "jermain", so the condition is false, and "false" will be printed.

1. Introducing 'elif':

name = "ahmed"

if name == "ahmed":

print("true")

elif name == "sara":

print("the name is sara")

else:

print("false")

Here, we introduce 'elif' (else if). It allows us to check multiple conditions. In this case, since the name is "ahmed", "true" will be printed.

1. 'elif' in Action:

name = "sara"

if name == "ahmed":

print("true")

elif name == "sara":

print("the name is sara")

else:

print("false")

Now, the name is "sara". The first condition is false, but the second condition (in the 'elif') is true, so "the name is sara" will be printed.

1. 'if-elif-else' with Unmatched Condition:

name = "mahmoud"

if name == "ahmed":

print("true")

elif name == "sara":

print("the name is sara")

else:

print("false")

In this final example, the name doesn't match any of the conditions in 'if' or 'elif'. Therefore, the code in the 'else' block is executed, and "false" is printed.

These examples demonstrate the power and flexibility of conditional statements in Python. They allow your code to make decisions and execute different blocks based on different conditions. Practice with these concepts, and you'll be able to create more complex and intelligent programs!

Extra code for lesson 30

section 6 lesson 30 python code

# using if statement

x = 30

y = 21

if x > y:

print("true")

# using if second time in numbers

if 5 < 10:

print("true")

# here the code does not give any resolt

if 5 < 3:

print("true")

# using else

x = 30

y = 21

if x < y:

print("true")

else:

print("false")

# using if and else with string

name = "ahmed"

if name == "ahmed":

print("true")

else:

print("false")

# changing the name to jermain

name = "jermain"

if name == "ahmed":

print("true")

else:

print("false")

# using elff

name = "ahmed"

if name == "ahmed":

print("true")

elif name == "sara":

print("the name is sara")

else:

print("false")

# changing the name to sara

name = "sara"

if name == "ahmed":

print("true")

elif name == "sara":

print("the name is sara")

else:

print("false")

# changing the name to mahmoud

# the third method for using if elff else

name = "mahmoud"

if name == "ahmed":

print("true")

elif name == "sara":

print("the name is sara")

else:

print("false")

Title: Section 6, Lesson 31: Nested Conditional Statements in Python

Introduction:

Good morning, esteemed students. I am Dr. Ahmed Sami, and today we will delve into an advanced topic in Python programming: nested conditional statements. This concept is crucial for creating complex decision-making structures in your code. Let's explore how we can use multiple layers of 'if' statements to create more sophisticated logic in our programs.

Body:

Let's examine the following code:

# using nesting in if statement

x = 5

if x > 2:

if x < 10:

if x == 5:

print("x is = 5 that is True")

Now, let's break this down line by line:

1. x = 5  
   Here, we're assigning the value 5 to the variable x. This is our starting point.
2. if x > 2:  
   This is our first conditional statement. It checks if x is greater than 2. Since x is 5, this condition is true, so we proceed to the next level of nesting.
3. if x < 10:  
   This is our second level of nesting. It checks if x is less than 10. Again, since x is 5, this condition is also true, so we continue to the next level.
4. if x == 5:  
   This is our third and innermost level of nesting. It checks if x is exactly equal to 5. In this case, it is, so we proceed to the next line.
5. print("x is = 5 that is True")  
   This line will only execute if all the above conditions are true. In our case, they are, so this message will be printed.

Let's consider some examples to understand this better:

Example 1: If we change x to 4, the output would be nothing because while the first two conditions are met, the innermost condition (x == 5) is not.

Example 2: If we change x to 11, there would be no output because the second condition (x < 10) is not met.

Example 3: If we change x to 1, there would be no output because the very first condition (x > 2) is not met.

This nested structure allows us to create very specific conditions that must be met in a particular order. It's like a series of gates, each one must be passed before reaching the next.

Conclusion:

Nested conditional statements are a powerful tool in Python programming. They allow us to create complex decision trees within our code, enabling more sophisticated logic and control flow. However, it's important to use them judiciously, as deeply nested conditions can make code harder to read and maintain. As you progress in your Python journey, you'll develop a sense for when and how to use these structures effectively.

Remember, the key to mastering nested conditionals is practice. I encourage you all to experiment with different conditions and nesting levels to truly understand their potential and limitations.

Extra python code for lesson 31

section 6 python code for lesson 31

# using nesting in if statement

x = 5

if x > 2:

if x < 10:

if x == 5:

print("x is = 5 that is True")

**Understanding Conditional Statements and Nested Conditionals in Python**

**Introduction**

Good morning, esteemed students. Today, we delve into a crucial aspect of Python programming: conditional statements. Mastering conditional statements is fundamental for creating logical flows in your programs. This lecture will cover the core concepts of 'if', 'elif', and 'else' statements, followed by an exploration of nested conditional statements. By the end of this session, you will have a comprehensive understanding of how to utilize these constructs effectively in your coding practices.

**Body**

**Basic Conditional Statements**

Let us begin with the simplest form of conditional statements, the 'if' statement.

x = 30

y = 21

if x > y:

print("true")

In this code snippet, we declare two variables, x and y, assigning them the values 30 and 21, respectively. The 'if' statement evaluates the condition . Since 30 is indeed greater than 21, the expression evaluates to true, and "true" is printed to the console.

**Using Literal Values in Conditions**

Next, we can also use literal values in our conditions:

if 5 < 10:

print("true")

Here, we check whether 5 is less than 10. As this condition holds true, the output will also be "true".

**Handling False Conditions**

It's essential to understand how Python handles false conditions:

if 5 < 3:

print("true")

In this case, since 5 is not less than 3, the condition evaluates to false, and nothing is printed as the code inside the 'if' block is skipped.

**Introducing 'else'**

Let us introduce the 'else' statement, which allows us to provide an alternative action when the 'if' condition is false:

x = 30

y = 21

if x < y:

print("true")

else:

print("false")

Here, since is false, the program executes the 'else' block, resulting in the output "false".

**'if-else' with String Comparisons**

String comparisons can also be utilized within conditional statements:

name = "ahmed"

if name == "ahmed":

print("true")

else:

print("false")

In this case, the name variable matches "ahmed", so "true" will be printed. Conversely, if we change the name:

name = "jermain"

if name == "ahmed":

print("true")

else:

print("false")

Now, the output will be "false" as the condition is not met.

**Introducing 'elif'**

The 'elif' statement expands our conditional capabilities by allowing us to test multiple conditions:

name = "ahmed"

if name == "ahmed":

print("true")

elif name == "sara":

print("the name is sara")

else:

print("false")

Since the name is "ahmed", it outputs "true". If we change the name to "sara", the output would then reflect that condition.

**'if-elif-else' with Unmatched Conditions**

Finally, consider the following example:

name = "mahmoud"

if name == "ahmed":

print("true")

elif name == "sara":

print("the name is sara")

else:

print("false")

Here, since the name "mahmoud" does not match any of the conditions, the output will be "false".

**Nested Conditional Statements**

Now, let us transition to nested conditional statements, which enable us to create more complex decision-making structures:

x = 5

if x > 2:

if x < 10:

if x == 5:

print("x is = 5 that is True")

This code defines a variable x with a value of 5. The first condition checks if , which is true. The second condition checks if , which is also true. Finally, the innermost condition checks if x == 5. Since all conditions are met, the program prints "x is = 5 that is True".

**Examples of Nested Conditions**

1. If x is changed to 4, there will be no output because the innermost condition fails.
2. If x is set to 11, the output will still be empty as the second condition is not satisfied.
3. Should x be 1, the output remains absent since the very first condition fails.

This nested structure creates a series of gates; each must be passed for the next to be evaluated.

**Conclusion**

In conclusion, conditional statements and nested conditionals are powerful tools in Python programming. They allow us to create complex decision trees within our code, enabling more sophisticated logic and control flow. However, it is crucial to apply these structures judiciously, as deeply nested conditions can complicate code readability and maintenance.

As you continue your journey in Python programming, I urge you to practice these concepts. Experimenting with various conditions and nesting levels will deepen your understanding of their potential and limitations. Remember, the key to mastering conditional statements lies in practice and experimentation. Thank you for your attention, and I look forward to our next session where we will explore further programming concepts.

Title: Section 6, Lesson 32: Logical Operators and Comparisons in Python

Introduction:

Good morning, esteemed students. I am Dr. Ahmed Sami, and today we'll be exploring logical operators and comparisons in Python. These are fundamental concepts that allow us to create complex conditions and make decisions in our code. Let's dive into each concept with examples.

Body:

1. Basic Comparison Operators:

# == : Checks if two values are equal

# > : Checks if the left value is greater than the right value

# < : Checks if the left value is less than the right value

# >= : Checks if the left value is greater than or equal to the right value

# <= : Checks if the left value is less than or equal to the right value

# != : Checks if two values are not equal

These operators are the building blocks of comparisons in Python. For example:

x = 5

y = 10

print(x == y) # False

print(x < y) # True

print(x >= 5) # True

print(x != y) # True

1. The 'and' Operator:

# and combines two conditions

x = 5

y = 10

if x > 2 and x < 10:

print("True")

Here, both conditions must be true for the code inside the 'if' statement to execute. In this case, it will print "True" because 5 is indeed greater than 2 and less than 10.

1. The 'or' Operator:

# or allows the code to execute if at least one of the conditions is true

x = 5

y = 10

if x > 2 or x <= 10:

print("true")

With 'or', only one condition needs to be true. This will print "true" because both conditions are actually true (5 > 2 and 5 <= 10).

1. The 'not' Operator:

# not inverts the condition, making it true if the original condition is false

x = 5

y = 10

if not(x == y):

print("false")

'not' reverses the boolean value. Here, it prints "false" because x is indeed not equal to y.

1. The 'in' Operator:

# in checks if an element is present within a sequence

name = "ahmed"

if "h" in name:

print("h in name")

'in' checks for membership. It will print "h in name" because 'h' is indeed in the string "ahmed".

1. The 'not in' Operator:

# not in checks if an element is not present within a sequence

name = "ahmed"

if "s" not in name:

print("not in")

'not in' is the opposite of 'in'. It prints "not in" because 's' is not in "ahmed".

1. The 'is' Operator:

# is checks if two variables point to the same object in memory

x = 5

y = 10

if x is y:

print("true")

'is' checks for identity, not equality. This won't print anything because x and y are different objects.

1. The 'is not' Operator:

# is not checks if two variables do not point to the same object in memory

x = 5

y = 10

if x is not y:

print("x is not y")

'is not' is the opposite of 'is'. This will print "x is not y" because x and y are indeed different objects.

Conclusion:

Today, we've explored various logical operators and comparisons in Python. These tools are essential for building complex logic in your programs and making precise comparisons. Remember, practice is key to mastering these concepts. I encourage you to experiment with these operators in your own projects to better understand how they work.

Thank you for your attention. Are there any questions about what we've discussed today?

Extra python code for lesson 32

section 6 python code for lesson 32

# == : Checks if two values are equal

# > : Checks if the left value is greater than the right value

# < : Checks if the left value is less than the right value

# >= : Checks if the left value is greater than or equal to the right value

# <= : Checks if the left value is less than or equal to the right value

# != : Checks if two values are not equal

# and combines two conditions

x = 5

y = 10

if x > 2 and x < 10:

print("True ")

# or allows the code to execute if at least one of the conditions is true

x = 5

y = 10

if x > 2 or x <= 10:

print("true")

# not inverts the condition, making it true if the original condition is false

# using not

x = 5

y = 10

if not(x ==y):

print("false")

# in checks if an element is present within a sequence

# using in

name = "ahmed"

if "h" in name:

print("h in name")

# not in checks if an element is not present within a sequence

# using not in

name = "ahmed"

if "s" not in name:

print("not in")

# is checks if two variables point to the same object in memory

# using is

x = 5

y = 10

if x is y:

print("true")

# is not checks if two variables do not point to the same object in memory

# using is not

x = 5

y = 10

if x is not y:

print("x is not y")